LC2 - LC4

Opto 22 FORTH

LC2: Release 2

LC4: Release 3

This technical document describes the features, specifications, and operations of the product.

The information in this manual has been carefully checked and is believed to be accurate; however, no responsibility is assumed for possible inaccuracies or omissions. Specifications are subject to change without notice.

Opto 22 warrants all its products to be free from defects in material or workmanship for 24 months from the manufacturing date code.

This warranty is limited to the original cost of the unit only and does not cover installation labor or any other contingent costs.

Table Of Contents

INTRODUCTION

The LC2/LC4 FORTH is a subset of the FORTH-83 standard with further enhancements which take advantage of LC2/LC4's hardware features. These features include accessing the real-time clock, the OPTOMUX serial port, the OPTOWARE driver and the PAMUX driver (LC4 only).

LC2/LC4 FORTH is a powerful and versatile language which, in the hands of knowledgeable programmers, can be used to create programs which, generally speaking, will execute faster and more efficiently than programs written using the LC2/LC4 BASIC. That is not to say, however, that LC2/LC4 BASIC is not useful. LC2/LC4 BASIC has advantages where speed and space are not critical but readability and ease of programming are important. Programmers who live by the FORTH philosophy may argue this point, and it is not our intent to proclaim which language is the best. Therefore, we offer both and leave the final decision to you; the end user. Neither is it our intent to teach the language of FORTH. That is something best left to the textbooks and the teachers. The Appendix contains some suggested reading material which may offer some help to beginning and advanced FORTH programmers. In addition, several examples are included which try to explain the additional Opto 22 FORTH Word Set and the interaction with the OPTOWARE driver and the PAMUX driver.

FEATURES

The major extensions to the LC2/LC4 FORTH are floating point numbers, an OPTOMUX driver and a PAMUX driver. The LC2 does not contain the PAMUX driver since it does not have the hardware needed to talk to the PAMUX boards.

FLOATING POINT NUMBERS

The new LC2/LC4 FORTH now supports floating point numbers and the associated transcendental functions (sine, cosine, etc.). The format for floating point numbers is as follows:

mantissa E exponent

When entered, the mantissa is a standard double number. The E, converts the mantissa to a floating point number and then inputs the exponent, a single precision number, and adjust the floating point number accordingly. The placement of the decimal point in the mantissa is significant, a 3.0 E 5 is not equal to a .30 E 5 (though a 3.0 is equal to a .30 in double precision numbers).

XON/XOFF PROTOCOL

The FORTH now supports the XON/XOFF protocol when downloading a program. The XON/XOFF protocol is enabled or disabled depending upon the value contained in the variable XON0.ENABLE. A zero value stored in XON0.ENABLE will disable the protocol, any other value will enable the protocol. If the LC2/LC4 FORTH seems to be working fine, but is printing strange characters at the beginning and end of each line, the XON/XOFF protocol is probably enabled.

NOTE: The XON/XOFF protocol will only function when you are downloading a program, not while you are running a program.

POWER UP MODES

When the controller is first powered up, it is in terminal mode. It will accept any new words at this point that are sent to it from the host port. It is important to note that the last word sent to the controller is the first word executed after power up when the AUTO jumper is installed. It is up to the user program to make forth words that allow starting and stopping the program from the host port. Once it is stopped, new words can be downloaded from any ASCII terminal or computer via the host port.

OPTOMUX AND PAMUX DRIVERS

The LC2/LC4 FORTH uses a structure for passing parameters to and from the OPTOMUX and PAMUX driver subroutines (the LC2 FORTH does not contain the PAMUX driver). Each driver uses the same defining word, PARAMETER-BLOCK, to define a structure for the subroutines. To use a particular PARAMETER-BLOCK in a call to the driver, you store its address in the variable PARAMETERS. The OPTOMUX driver (OPTOWARE) expects up to six items in the defined PARAMETER-BLOCK. The first three items act like-single precision variables and the second three items act like single-precision arrays.

ERRORS	**returned error number (variable)**
ADDRESS	**OPTOMUX board address (variable)**
COMMAND	**command to perform (variable)**
POSITIONS	**list of modules to use (16 element array)**
MODIFIERS	**command modifiers (2 element array)**
INFO	**data (16 element array)**

The following code demonstrates how to use the PARAMETER-BLOCK structure:

```
PARAMETER-BLOCK RONNY-BABY      \ define a parameter block
RONNY-BABY PARAMETERS !!        \ set the address so we can use it
243 ADDRESS !                   \ set the OPTOMUX board address
0 COMMAND !                     \ lets do a power-up clear command
OPTOWARE                        \ and call the driver
12 COMMAND !                    \ lets do a get relay status command
OPTOWARE                        \ and call the driver again
3 INFO ?                        \ print the status of module 3
```

You are allowed to have multiple PARAMETER-BLOCK defined at one time, however, only one is active at a time (the one whose address is in the variable PARAMETERS).

The PAMUX driver uses the same defining word, PARAMETER-BLOCK, but uses some different elements:

ERRORS	**returned error number (variable)**
ADDRESS	**OPTOMUX board address (variable)**
COMMAND	**command to perform (variable)**
POSITION	**module to use (variable)**
INFO	**data (8 element array)**

NOTE: The OPTOMUX driver uses the array call POSITIONS and the PAMUX driver uses a variable call POSITION (no S).

LC2 FORTH QUICK REFERENCE GUIDE

ARITHMETIC

*	w1 w2 -- w3	Multiply
*/	n1 n2 n3 -- n4	n4 = (n1*n2)/n3
*/MOD	n1 n2 n3 -- rem quot	*/ with remainder
*D	n1 n2 -- d	Single-to-double multiply
+	w1 w2 -- w3	Add
+!	w1 addr --	Add w to number at addr
+-	w1 w2 -- w3	If w2<0 make w1 negative
-	w1 w2 -- w3	Subtract (w1-w2)
/	n1 n2 -- n3	Divide n1 by n2
/MOD	n1 n2 -- rem quot	Divide with remainder
1+	w1 --w1	Increment by one
1-	w1 -- w1	Decrement by one
2*	n1 -- n2	Multiply by two
2+	w1 -- w2	Increment by two
2-	w1 -- w2	Decrement by two
2/	n1 -- n2	Signed divide by two
ABS	n -- u	Absolute value of single
D+	wd1 wd2 -- wd3	Double-precision add
D-	wd1 wd2 -- wd3	Double-precision subtract
D+-	d1 n -- d2	Double precision +-
D2*	d1 -- d2	Multiply d1 by 2
DABS	d -- ud	Absolute value of double
DMAX	d1 d2 -- d3	Leave larger, double number
DMIN	d1 d2 -- d3	Leave smaller, double number
DNEGATE	d1 -- d2	Change sign of double
DSHIFT	wd1 n -- wd2	Shift wd1 by n
F*	float1 float2 -- float3	Multiply float1 by float2
F+	float1 float2 -- float3	Floating point add
F-	float1 float2 -- float3	Subtract float2 from float1
F/	float1 float2 -- float3	Divide float1 by float2
FNEGATE	float1 -- float2	Change sign of a floating point number
FPERR	-- n	Floating point error variable
FSIN	float1 -- float2	Return sine of float1 in radians
FSQRT	float1 -- float2	Return square root of float1
F^	float1 float2 -- float3	Raise float1 to the float2 power
M/MOD	d n -- rem quot	Mixed divide with remainder
MAX	n1 n2 -- n3	Leave larger number
MIN	n1 n2 -- n3	Leave smaller number
MOD	n1 n2 -- n3	Leave remainder of n1/n2
MU/MOD	ud1 u1 -- rem quot	Mixed divide with remainder
NEGATE	n1 -- n2	Change sign
S>D	n -- d	Sign extend to double
SHIFT	w1 n -- w2	Shift w1 by n
UM*	u1 u2 -- ud	Unsigned mixed multiply
UM/MOD	ud u1 -- u2 u3	Unsigned mixed divide

LOGICAL

AND	16b1 16b2 -- 16d3	Bitwise logical AND
FALSE	-- 0	Constant
NOT	16b1 -- 16b2	One's complement
OR	16b1 16b2 -- 16b3	Bitwise logical OR
TOGGLE	addr byte --	XOR byte at addr
TRUE	-- -1	Constant
XOR	16b1 16b2 -- 16b3	Bitwise exclusive OR

COMPARISON

0<	n -- flag	True if n < zero
0=	w -- flag	True if w = zero
0>	n -- flag	True if n > zero
0<>	n -- flag	True if n not equal to 0
<	n1 n2 -- flag	True if n1 < n2
<>	n1 n2 -- flag	True if n1 not = n2
=	w1 w2 -- flag	True if w1 = w2
<	n1 n2 -- flag	True if n1 > n2
?CONDITION	flag --	ABORT "conditionals wrong" if false
AWAKE	n --	Wake up sleeping LC2
	--	
D0=	wd -- flag	True if wd = 0
D<	d1 d2 -- flag	True if d1 < d2
D=	wd1 wd2 -- flag	True if wd1 = wd2
D>	d1 d2 -- flag	True if d1 > d2
DU<	ud1 ud2 -- flag	True if d1 < d2 (unsigned)
DU>	d1 d2 -- flag	True if d1 > d2 (unsigned)
F0=	float -- flag	True if float = 0
F<	float1 float2 -- flag	True if float1 < float2
F=	float1 float2 -- flag	True if float1 = float2
F>	float1 float2 -- flag	True if float1 > float2
OF	-- addr n	(compiling)
	n1 n2 -- n1	(if no match was found)
	n1 n2 --	(if match was found)
U<	u1 u2 -- flag	True if u1 < u2 (unsigned)
U>	u1 u2 -- flag	True if u1 > u2 (unsigned)

MEMORY

!	16b addr --	Store 16B at addr
2!	32b addr --	Store d at addr
2@	addr -- 32b	Fetch double number
?	addr --	Print number at addr
@	addr --16b	Fetch 16b
@ @	addr -- 16b	Indirect fetch from memory
ADDRESS	-- addr	Address field in parameter block
BANK.!	16b addr --	Store 16b at addr
BANK.2!	32b addr --	Store d at addr
BANK.2@	addr -- 32b	Fetch double number
BANK.@	addr -- 16b	Fetch 16b
BANK.C!	16b addr --	Store lower byte at addr
BANK.C@	addr -- 16b	Store lower byte at addr
C!	16b addr --	Store lower byte at addr
C@	addr -- 16b	Store lower byte at addr
CMOVE	addr1 addr2 u --	Move byte string LO -- HI
CMOVE	addr1 addr2 u --	Move byte string HI -- LO
ERRORS	-- addr	Errors field in parameter-block
F!	float addr --	Store floating point at addr
F@	addr -- float	Fetch float at addr
FILL	addr u 8b --	Fill memory with byte
INFO	n -- addr	Info-array in parameter-block
MODIFIERS	n -- addr	Modifier array in parameter-block
PARAMETERS	-- addr	Point to current parameter-block
POSITIONS	n -- addr	Position-array in parameter-block
TOGGLE	addr byte --	XOR byte at addr

CONVERSION

#	+d1 -- +d2	Binary digit to ASCII
#>	32b -- addr +n	End ASCII conversion
#S	+d -- 00	Convert rest of digits
<#	--	Start ASCII conversion
BASIC	--	Exit FORTH and go to BASIC
BODY>	addr1 -- addr2	Convert PFA to CFA
CONVERT	+d addr1 -- +d2 addr 2	ASCII string to binary
D>F	d -- float	
DECIMAL	--	BASE=DECIMAL
DIGIT	char n -- n2 flag (valid digit) char n -- flag (invalid digit)	ASCII to binary base n
E	d -- float	Convert a double-precision number to float
F>D	float -- d	Convert a float to a double
FCOS	float1 -- float2	Return cosine of float1 in radians
FEXP	float1 -- float2	Calculate "e" raised to power float1
FLN	float1 -- float2	Natural log of float1 (float2)
HEX	--	BASE = Hexadecimal
HLD	-- addr	Addr of latest conversion char
HOLD	char --	Insert char into string
S>D	n -- d	Converts a single-precision number to double-precision
SIGN	n --	Insert sign into string

STRING

-TRAILING	addr +n2 -- addr +n2	suppress trailing blanks
BL	-- 32 (decimal) -- 20 (hex)	Put ASCII "space" on stack
COUNT	addr1 -- addr2 +n	Get count from string
ENCLOSE	addr1 char -- addr1 n1 n2 n3	Text parsing primitive
WORD	char -- addr	Parse input string

INPUT/OUTPUT

#IN	-- n	Input n from selected port
.	n --	Print signed number
."	--	Print string up to "
.(	--	Print string up to)
2#IN	-- d	ASCII input to double
CR	--	Output CRLF sequence
D.	d --	Print double number
D.R	d width --	Print a double-precision number
D>R	d width --	Print right justified
DU.	ud --	Print unsigned, double number
DU.R	ud n --	Print unsigned, double, right-justified
DUMP	addr n --	Display contents of n bytes of memory
EMIT	16b --	Output one character
EXPECT	addr +n --	String input stored at addr
F.	float --	Print a floating-point number
F.R	float n --	Print a floating-point number (right-justified)
F?	addr --	Print float stored at addr
HOURS!	n --	Set the hour on the real time clock
HOURS@	-- n	Read the hour on the real time clock
KEY	--- 16b	Input a character from port
LOC	-- n	Number of bytes in serial port buffer
LOF	-- n	Number of free bytes in serial port buffer
P!	n port --	Output a byte to I/O port
P@	port -- n	Input a byte from I/O port
QUERY	--	Input string to TIB
SPACE	--	Print one space
SPACES	+n --	Print n spaces
TYPE	addr +n --	Print character string
U.	u --	Print unsigned number
U.R	u width --	Print unsigned number (right justified)
U>R	u width --	Print unsigned (right justified)

STACK

-ROT	16b1 16b2 16b3 -- 16b3 16b1 16b2	Rotate 2nd number to top
-2ROT	32b1 32b2 32b3 -- 32b3 32b1 32b2	Rotate 2nd number to top
-3ROT	48b1 48b2 48b3 -- 48b3 48b1 48b2	Rotate 2nd triple number to top
2DROP	32b --	Discard double number
2DUP	32b -- 32b 32b	Duplicate double number
2OVER	32b1 32b2 -- 32b1 32b2 32b1	Copy 2nd double to top
2ROT	32b1 32b2 32b3 -- 32b2 32b3 32b1	Rotate 3rd double number to top
2SWAP	32b1 32b2 -- 32b2 32b1	Reverse top two doubles
3DROP	48b1 --	Discard one triple-precision number
3DUP	48b1 48b1 48b1 --	Duplicate top triple number
3ROT	48b1 48b2 48b3 -- 48b2 48b3 48b1	Rotate 3rd triple number to top
3SWAP	48b1 48b2 -- 48b2 48b1	Reverse top two triple numbers
>R	16b --	Move 16b to return stack
?DUP	16b -- 16b 16b or 0 -- 0	Duplicate 16b if not zero
DEPTH	-- +n	# of items on stack
DROP	16b --	Discard single number
DUP	16b -- 16b 16b	Duplicate single number
OVER	16b1 16b2 -- 16b1 1b62 16b1	Copy 2nd item to top
PICK	+n --- 16b	Copy nth item to top
R<	-- 16b	Pop the return stack
R@	-- 16b	Copy from return stack
ROLL	+n	Rotate nth item to top
SWAP	16b1 16b2 -- 16b2 16b1	Exchange top two numbers
ROT	16b1 16b2 16b3 -- 16b2 16b3 16b1	Rotate 3rd item to top
ROT	16b1 16b2 16b3 -- 16b3 16b1 16b2	Rotate top item to 3rd

CONTROL

+LOOP	n --	Increment loop index by n
<MARK	-- addr	Fetch current dictionary pointer
<RESOLVE	addr --	Compile word into current definition
>MARK	-- addr	Fetch addresses of next free location
>RESOLVE	addr --	Place current dictionary pointer at addr
?<MARK	-- flag addr	Puts true on stack and does <MARK
?<RESOLVE	flag -- addr	Perform a <RESOLVE if true
?>MARK	-- flag addr	Puts true on stack and does >MARK
?>RESOLVE	flag addr --	Perform a >RESOLVE if true
?BRANCH	flag --	Perform branch if true
?DO	w1 w2 --	Perform DO.LOOP if w1 <>w2
?LEAVE	flag --	Execute LEAVE if true
AGAIN	--	Branch to word after BEGIN
BEGIN	--	Marks start of BEGIN loop
CASE	--- (executing) addr n --- (compiling)	Start of CASE control structure
DO	limit start --	Loop from start to start + limit
ELSE	--	Conditional part of IF
ENDCASE	n --- (if no match) -- (if matched)	End of CASE control structure
EXECUTE	addr --	Execute word at addr
EXIT	--	Exit from colon definition
IF	flag --	Conditional execution
LEAVE	--	Jump out of DO loop
LOOP	--	Increment loop index by 1
QUIT	--	Return control to outer interpreter
RECURSE	--	Execute recursively
REPEAT	--	Branch to word after BEGIN
THEN	--	Conditional part of IF
UNTIL	flag --	Marks the end of BEGIN..UNTIL loop
WHILE	flag --	Conditional execution

COMPILER WORD

,	-- addr	Compilation address
(	--	Start a comment
(;CODE)	--	Run time procedure for ;CODE
(.")	addr -- addr	Run time procedure for ."
(+LOOP)	n --	Run time procedure for +LOOP
(?DO)	--	Run time procedure for ?DO
(?LEAVE)	--	Run time procedure for ?LEAVE
(DO)	--	Run time procedure for DO
(LEAVE)	--	Run time procedure for LEAVE
(LIT)	--	Run time procedure for LIT
(LOOP)	--	Run time procedure for LOOP
,	16b --	Store 16b at HERE
:	--	Start colon definition
;	--	End colon definition
;CODE	--	Stops compiling and executes ASSEMBLER
	sys1 -- sys2	
>LINK	addr1 -- addr2	Get CFA's corresponding LFA
>NAME	addr1 -- addr2	Get CFA's corresponding NFA
ASCII	--	Put an ASCII character onto stack
BRANCH	--	Unconditional branch
C,	byte --	Store byte at HERE
COMPILE	--	Compile a compilation address
CREATE	--	Build a dictionary header
DLITERAL	--32b	Compile a double number
	32b --	
DOES>	-- addr	Define a run time action
ENDOF	addr n1 -- addr2 n2	Transfer control to word after ENDCASE
NAME>	addr1 -- addr2	Get NFA's corresponding CFA
[	--	Set interpret state
[']	-- addr	Compile the compilation addr
[COMPILE]	--	Compile the following word
]	--	Set Compilation state

DEFINING WORD

2ARRAY	n --	Create a double-precision array
2CONSTANT	32b --	Create double constant
2VARIABLE	--	Create double variable
ARRAY	n --	Create a single-precision array
CONSTANT	16b --	Create a constant
FARRAY	n --	Create a float array of n elements
FCONSTANT	float --	Create a floating point constant
FVARIABLE	--	Create a floating point variable
PARAMETER-BLOCK		--Structure of variables for passing to driver
USER	n --	Create a user variable
VARIABLE	--	Create a variable
VOCABULARY	--	Create a vocabulary

SYSTEM

!CSP	--	Store parameter stack addr
#TIB	-- addr	Addr of TIB string length
(FIND)	addr1 addr2 -- addr3 n	Searches current directory
.CPU	--	Print processor name
.NAME	addr --	Print name in dictionary
.S	--	Print the stack
>BODY	addr1 -- addr2	Get CFA's corresponding PFA
>IN	-- addr	Input stream pointer
?COMP	--	Check if compiling
?CSP	--	Check parameter stack pointer
?EXEC	--	Check if executing
?PAIRS	16b1 16b2 --	Execute abort if 16b <> 16b
?STACK	--	Check limits of stack pointer
?KEY	-- flag	True if character received
ABORT	n..n --	Clear stack and quit
ABORT"	flag --	Conditional abort
ALLOT	w --	Allocates w bytes in the dictionary
BASE	--	System radix
C/L	-- n	Constant leaving # of characters per line
CLEAR.BUF	--	Reset communication buffers
CLOCK.TICK	-- addr	Variable incremented by RTC
COLD	--	Restart FORTH
COM1	--	Select Optomux port for serial I/O
COM2	--	Select serial port 2 on LC4 for serial I/O
COM3	--	Select serial port 3 on LC4 for serial I/O
CONSOLE	--	Select host port for serial I/O
CONTEXT	-- addr	First vocabulary to search
CSP	-- addr	User variable for stack pointer
CURRENT	-- addr	User variable for vocabulary pointer
DEFINITIONS	--	Select compilation vocabulary

DP	-- addr	Next dictionary location
DPL	-- addr	# of digits right of decimal
EMPTY	--	Remove all user compiled FORTH words
ERROR	addr1 n1 -- addr2 n2	Print a system error
EVEN	-- n	Constant for even parity
FENCE	-- addr	FORGET boundary
FIND	addr1 -- addr2 n	Search name in dictionaries
FORGET	--	Remove from dictionary
FORTH	--	Select FORTH vocabulary
FORTH-83	--	Verify FORTH-83 system
HERE	-- addr	Next available dictionary addr
I	-- w	Inner loop index
IMMEDIATE	--	Mark latest definition
INTERPRET	--	Outer text interpreter
J	-- w	Inner loop index
K	-- w	Inner loop index
LATEST	-- addr	Most recently compiled definition
LC.ADDRESS	-- addr	LC2's address for SLEEP/AWAKE
LITERAL	-- 16b	Compile a number
	16b --	
NEXT-LINK	-- addr	Addr of NEXT
NO	-- n	Constant for no parity
NOOP	--	Does nothing
OPTOWARE	--	Calls Optoware driver
ODD	-- n	Constant for odd parity
OUT	-- addr	Counts number of EMITS
PAD	-- addr	System scratch pad
PAMWARE	--	Calls the Pamux driver
R0	-- addr	Initial addr of return stack
RP!	--	Initializes return stack
RP@	-- addr	Addr of return stack
SET.SERIAL	n1 n2 n3 -- flag	Sets serial port parameters
S0	-- addr	Initial addr of parameter stack
SMUDGE	--	Toggles smudge bit
SPAN	-- addr	# of chars received by EXPECT
SP!	--	Initializes parameter stack
SP@	-- addr	Addr of parameter stack
STATE	-- addr	Compilation state
TASK	--	Boundary marker
TIB	-- addr	Text input bugger
TRAVERSE	addr1 n -- addr2	Move across name field
TX.ENABLE	-- addr	State of com port transmitter
VOC-LINK	-- addr	Addr of vocabulary link field
WIDTH	-- addr	MAX # of chars in name field
WORDS	--	Lists names in vocabulary
XON0.ENABLE	-- addr	Com0 xon/xoff enable variable
XON1.ENABLE	-- addr	Com1 xon/xoff enable variable
XON2.ENABLE	-- addr	Com2 xon/xoff enable variable
XON3.ENABLE	-- addr	Com3 xon/xoff enable variable
\	--	Start of comment

LC2/LC4

ADDRESS	-- addr	Address field in PARAMETER-BLOCK
AWAKE	n --	Wake up sleeping LC2
CLOCK.TICK	-- addr	Counter variable
COMMAND	-- addr	Command field in PARAMETER-BLOCK
DATE!	-- n1 n2 n3 --	Set date on RTC
DATE@	-- n1 n2 n3	Fetch date from RTC
DAYS!	n --	Set day on RTC
DAYS@	-- n	Fetch day from RTC
ERRORS	-- addr	Errors field in PARAMETER-BLOCK
HOURS!	n --	Set hours of RTC
HOURS@	-- n	Fetch hours from RTC
INFO	n -- addr	Info-array in PARAMETER-BLOCK
LC.ADDRESS	n --	LC2's address for SLEEP/AWAKE
MINUTES!	n --	Set minutes on RTC
MINUTES@	-- n	Fetch minutes from RTC
MODIFIERS	n -- addr	Modifier-array in PARAMETER-BLOCK
MONTHS!	n --	Set months on RTC
MONTHS@	-- n	Fetch months from RTC
OPTOWARE	--	Call OPTOMUX™ driver
PAMWARE	--	Call PAMUX™ driver
PARAMETER-BLOCK		--Parameter-block for OPTOWARE™
PARAMETERS	-- addr	Point to current parameter-block
POINTER	addr --	Create memory pointer
POSITION	-- addr	Position variable in parameter-block
POSITIONS	n -- addr	Positions-array in parameter-block
SECONDS!	n --	Set seconds on RTC
SECONDS@	-- n	Fetch seconds from RTC
SLEEP	n --	Put LC2 to sleep
TIME!	n1 n2 n3 --	Set time on RTC
TIME@	-- n1 n2 n3	Fetch time from RTC
YEARS!	n --	Set year on RTC
YEARS@	-- n	Fetch year from RTC

GLOSSARY

The major parts included in the Glossary are: an explanation of the conventions used throughout the manual, an alphabetized Standard Word Set for quick reference and the actual Word Set Glossary,

GLOSSARY CONVENTIONS

Glossary Word Definition Layout

(WORD DEFINITION)
"natural language pronunciation"

STACK:
(Stack Notation)

CATEGORY:
(Word Attribute) -(Category)

DESCRIPTION:
(Word Description)

Word Definitions

All LC2 FORTH words are defined in upper case. In order to be executed, they must be called in upper case. User defined words can be defined in either case, but must be called in the form (upper or lower case) which they were defined. For ease of verbal communication, the natural language pronunciation (as per FORTH-83 Standards) is surrounded by double quotes where necessary.

Text Input

Word definitions may include the following two symbols to represent text input. This notation refers to text from the input stream, not to values from the data stack:

<name>

An arbitrary FORTH word accepted from the input stream:

ccc

A sequence of arbitrary chracters accepted from the input stream. The number of characters accepted may be from 0 to 255 and are accepted until the occurrence of a specified delimiter.

Word Attributes

Each word definition listed may contain certain capitalized symbols which indicate particular attributes of the defined word. These attributes are described below:

C	The word may only be used within a colon definition.
I	Immediate, will execute during compilation.
U	User variable.
83	Part of the FORTH 83 Required Word Set.
79	Unchanged from the FORTH 79 Required Word Set.

Word Descriptions

Each word in the glossary is followed by a brief description and an example to better illustrate its usage. The descriptions are reproduced from the FORTH-83 standards document and modified only where exceptions are taken or, for the sake of clarity, a better description is required.

Stack Notation

The following symbolic diagram shows the effect on the stack when a FORTH word is executed:

before -- after

Two dashes separate the contents of the stack before and after execution. The elements to the left of the dashes represent the contents of the stack before execution. The elements to the right of the dashes represent the contents of the stack after the word is executed. The top of the stack is always to the right. For example, if the stack contains the numbers 1,2 and 3, with 3 being on the top of the stack, the diagram for the word ROT can be shown as:

1 2 3 – 2 3 1

Stack Abbreviations

The following is a list of stack parameter abbreviations used throughout the LC2 FORTH manual. These abbreviations are consistent with those used in the FORTH-83 Standards document with minor exceptions. Suffixes are used to differentiate between multiple parameters of the same type.

Stack Abbr.	Number Type	Range (Decimal)	Minimum Field
addr	16-bit memory address	0..65,535	16
byte	8-bit byte (represented as a 16-bit number with the high order 8-bits equal to zero)	0..255	8
char	character (same representation as byte)	0..255	8
flag	boolean	0=false, -1=true	16
true	boolean	-1 as a result	16
false	boolean	0	16
b	bit 0..1	1	
8b	8 arbitrary bits (byte)	not applicable	8
16b	16 arbitrary bits (word)	not applicable	16
32b	32 arbitrary bits (double word)	not applicable	32
n	integer number (signed, 16 bits)	-32,768..32,767	16
+n	positive integer	0..32,767	16
d	double-precision integer (signed, 32-bits, high order bits on top of stack)	-2,147,483,648.. 2,147,483,647	32
+d	positive, double-precision integer	0..2,147,483,647	32
u	unsigned integer	0..65,535	16
w	unspecified integer (signed or unsigned)	-32,768..65,535	16
ud	unsigned, double integer	-..4,294,967,295	32
wd	unspecified, double integer	-2,147,483,648.. 4,294,967,295	32
sys	Temporary system data	not applicable	na

Categories of FORTH Words

Arithmetic	Performs an arithmetic or bit manipulation operation.
Comparison	Performs a comparison, then returns a flag to the stack Flags are boolean; -1 = true and 0 = false.
Compiler word	Has both compile-time and run-time action.
Control	Causes conditional or unconditional branching of program control at execution time.
Conversion	Converts ASCII data to binary or vice-versa.
Defining word	Creates a dictionary definition from the input stream.
Input	Reads data from host, OPTOMUX or CPU ports.
Logical	Performs a logical operation such as AND, OR, XOR or NOT.
Memory	Reads or changes bytes in memory.
Output	Controls or transfers data to the host, OPTOMUX, or CPU ports.
Stack	Changes the number of items or the position of items on the stack.
String	Performs operations on byte strings which may include parsing, counting or comparisons.
System	Provides access to the FORTH system.

STANDARD WORD SET

The LC2/LC4 FORTH Dictionary is composed of the majority of the FORTH-83 Required Word Set. Words not included are some of the file-related words in the device layer of the FORTH-83 Required Word Set. The LC2/LC4 FORTH Dictionary also includes words from the FORTH-83 Double Number Extension Word Set and the Controlled Reference Word Set. For more information on the FORTH-83 Standard, contact the FORTH Standards Team listed in the Appendix. An enhanced word set which includes LC2/LC4 and OPTOWARE™ specific words has also been added to makeup the complete LC2/LC4 FORTH Dictionary.

The following is a list of the LC2/LC4 FORTH Standard Word Set. Refer to the next section for information on the LC2/LC4 FORTH enhanced word set.

!	.NAME	<>	AWAKE
!CSP	.S	<MARK	BANK.!
#	/	<RESOLVE	BANK.2!
#>	/MOD	=	BANK.2@
#IN	0	>	BANK.@
#S	0<	>BODY	BANK.C!
#TIB	0<>	>IN	BANK.C@
'	0=	>LINK	BASE
(	0>	>MARK	BEGIN
(.")	1	>NAME	BL
(+LOOP) (;CODE)	1+	>R	BODY>
(?DO)	1-	>RESOLVE	BRANCH
(?LEAVE)	2	?	C!
(DO)	2!	?<MARK	C,
(FIND)	2#IN	?<RESOLVE	C/L
(LEAVE)	2*	?>MARK	CASE
(LIT)	2+	?>RESOLVE	C@
(LOOP)	2-	?BRANCH	CLEAR.BUF
*	2/	?COMP	CLOCK.TICK
*/	2@	?CONDITION	CMOVE
*/MOD	2ARRAY	?CSP	CMOVE>
*D	2CONSTANT	?DO	COLD
+	2DROP	?DUP	COM1
+!	2DUP	?EXEC	COM2 (LC4)
+-	2OVER	?KEY	COM3 (LC4)
+LOOP	2ROT	?LEAVE	COMMAND
,	2SWAP	?PAIRS	COMPILE
-	2VARIABLE	?STACK	CONSOLE
-1	3	@	CONSTANT
-2	3DROP	@@	CONTEXT
-2ROT	3DUP	ABORT	CONVERT
-3	3ROT	ABORT"	COUNT
-3ROT	3SWAP	ABS	CR
-4	4	ADDRESS	CREATE
-ROT	:	AGAIN	CSP
-TRAILING	;	ALLOT	CURRENT
.	<	AND	D+
."	<#	ARRAY	D-
.(		ASCII	D+-

(LC4): This word only available with LC4

D.
D.R
D0=
D2*
D<
D=
D>
D>F
DABS
DATE!
DATE@
DAYS!
DECIMAL
DEFINITIONS
DEPTH
DIGIT
DLITERAL
DMAX
DMIN
DNEGATE
DO
DOES>
DP
DPL
DROP
DSHIFT
DU.
DU.R
DU<
DU>
DUMP
DUP
E
ELSE
EMIT
EMPTY
ENCLOSE
ENDCASE
ENDOF
ERROR
ERRORS
EVEN
EXECUTE
EXIT
EXPECT
F!
F*
F+
F-
F.
F.R
F/
F0=
F<
F=
F>
F>D
F?
F@
FALSE
FARRAY
FCONSTANT
FCOSINE
FCOS
FENCE
FEXP
FILL
FIND
FLN
FNEGATE
FORGET
FORTH
FORTH-83
FPERR
FSIN
FSQRT
FVARIABLE
F^
HERE
HEX
HLD
HOLD
HOURS!
HOURS@
I
IF
IMMEDIATE
INFO
INTERPRET
J
K
KEY
LATEST
LC.ADDRESS
LEAVE
LOC
LITERAL
LOF
ROT
RP!
RP@
S0
S>D
SECONDS!
LOOP
M/MOD
MAX
MIN
MINUTES!
MINUTES@
MOD
MODIFIERS
MONTHS!
MONTHS@
MU/MOD
NAME>
NEGATE
NEXT-LINK
NO
NOOP
NOT
ODD
OF
OLD
OPTOWARE
OR
OUT
OVER
P!
P@
PAD
PAMWARE (LC4)
PARAMETER-BLOC
KPARAMETERS
PICK
POINTER
POSITION (LC4)
POSITIONS
QUERY
QUIT
R>
R@>
R0
RECURSE
REPEAT
ROLL
ROT
RP!
RP@
S0
S>D
SECONDS!
SECONDS@
SET.SERIAL
SHIFT
SIGN
SLEEP
SMUDGE
SPACE
SPACES
SPAN
SP!
SP@
STATE
SWAP

TASK
THEN
TIB
TIME!
TIME@
TOGGLE
TRAVERSE
TRUE
TX.ENABLE
TYPE
U.
U.R
U<
U>
UM*
UM/MOD
UNTIL
USER
VARIABLE
VOC-LINK
VOCABULARY
WHILE
WIDTH
WORD
WORDS
XON0.ENABLE
XON1.ENABLE
XON2.ENABLE-LC
4XON3.ENABLE-L
C4XOR
YEARS!
YEARS@
[
[']
[COMPILE]
]
/

(LC4): This word only available with LC4

WORD SET GLOSSARY

!
"store"

STACK:

16b addr --

CATEGORY:

79 - Memory

DESCRIPTION:

16b is stored at addr.

!CSP
"store-c-s-p"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

!CSP stores the current parameter stack address in the user variable CSP. This word is used in conjunction with ?CSP to determine if the parameter stack has changed after compilation.

#
"sharp"

STACK:

+d1 -- +d2

CATEGORY:

79 - Conversion

DESCRIPTION:

The remainder of +d1 divided by the value of BASE is converted to an ASCII character and appended to the output string toward lower memory addresses. +d2 is the quotient and is maintained for further processing. Typically used between <# and #>.

#>
"sharp-greater"

STACK:

32b -- addr +n

CATEGORY:

79 - Conversion

DESCRIPTION:

Pictured, numeric output conversion is ended dropping 32b. addr is the address of the resulting output string. +n is the number of characters in the output string. addr and +n together are suitable for TYPE.

#IN
"number-in"

STACK:

-- n

CATEGORY:

- Input, Conversion

DESCRIPTION:

#IN converts numeric characters being received at the selected input port into a single-precision number and places this number on the stack. Conversion stops if a non-numeric character is received. If the number is larger than a single-precision value, the upper bits of the result will be incorrect.

#S
"sharp-s"

STACK:

+d -- 00

CATEGORY:

79 - Conversion

DESCRIPTION:

+d is converted, appending each resultant character into the pictured numeric output string until the quotient (see: #) is zero. A single zero is added to the ouput string if the number was initially zero. Typically used between <# and #>.

#TIB
"sharp-t-i-b"

STACK:

-- addr

CATEGORY:

U,83 - System

DESCRIPTION:

#TIB is the address of a variable containing the number of bytes in the text input buffer. #TIB is accessed by WORD when BLK is zero. {{0..capacity of TIB}}

'
"tick"

STACK:

-- addr

CATEGORY:

83 - Compiler word

DESCRIPTION:

Used in the form: ' <name>

addr is the compilation address of <name>. An error condition exists if <name> is not found in the currently active search order.

(
"paren"

STACK:

--

-- (compiling)

CATEGORY:

I,83 - Compiler word

DESCRIPTION:

Used in the form: (ccc)

The characters ccc, delimited by) (closing parenthesis), are considered comments. Comments are not otherwise processed. The blank following (is not part of ccc. (may be freely used while interpreting or compiling. The number of characters in ccc may be from zero to the number of characters remaining in the input stream up to the closing parenthesis.

(.")
"paren-dot-quote"

STACK:

addr -- addr1

CATEGORY:

- Compiler word

DESCRIPTION:

(.") is a run-time procedure compiled by ." and is used to output the text string compiled into the dictionary by ." . addr is the address of the length byte of the text string. addr1 is the address of the next word to interpret.

(+LOOP)
"paren-plus-loop"

STACK:

n --

CATEGORY:

- Compiler word

DESCRIPTION:

(+LOOP) is the run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion.

(;CODE)
"paren semi-colon code"

STACK:

--

CATEGORY:

-Compiler word

DESCRIPTION:

(;CODE) is the run-time procedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence.

(?DO)
"paren-query-do"

STACK:

--

CATEGORY:

- Compiler word

DESCRIPTION:

(?DO) is the run-time procedure compiled by ?DO to indicate the start of an iterative loop.

(?LEAVE)
"paren-query-leave"

STACK:

--

CATEGORY:

-Compiler word

DESCRIPTION:

(?LEAVE) is the run-time procedure compiled by ?LEAVE.

(DO)
"paren-do"

STACK:

--

CATEGORY:

- Compiler word

DESCRIPTION:

(DO) is the run-time procedure compiled by DO to move the loop control parameters to the return stack. See DO.

(FIND)
"paren-find"

STACK:

addr1 addr2 -- pfa b tf (successful)
addr1 addr2 -- ff (unsuccessful)

CATEGORY:

- System

DESCRIPTION:

(FIND) will perform a dictionary search starting at the name-field address addr2, and try to match the text at address addr1. If successful, the parameter field address (pfa), the length byte of the name-field (b), and a boolean true flag (tf) will be returned on the stack. If unsuccessful, a boolean false flag (ff) will be returned.

(LEAVE)
"paren-leave"

STACK:

--

CATEGORY:

- Compiler word

DESCRIPTION:

(LEAVE) is the run-time procedure which is compiled by LEAVE to transfer execution to just beyond the next LOOP or +LOOP. See LEAVE.

(LIT)
"paren-lit"

STACK:

--

CATEGORY:

- Compiler word

DESCRIPTION:

(LIT) is the run-time procedure which is compiled by LIT to place the 16-bit contents of the next dictionary location onto the parameter stack.

(LOOP)
"paren-loop"

STACK:

--

CATEGORY:

- Compiler Word

DESCRIPTION:

(LOOP) is the run-time procedure compiled by LOOP which increments the loop index and tests for a loop completion.

*
"times"

STACK:

w1 w2 -- w3

CATEGORY:

79 - Arithmetic

DESCRIPTION:

w3 is the least-significant 16-bits of the arithmetic product of w1 times w2.

*/
"times-divide"

STACK:

n1 n2 n3 -- n4

CATEGORY:

83 - Arithmetic

DESCRIPTION:

n1 is first multiplied by n2 producing an intermediate 32-bit result. n4 is the floor of the quotient of the intermediate 32-bit result divided by the divisor n3. The product of n1 times n2 is maintained as an intermediate 32-bit result for greater precision than the otherwise equivalent sequence: n1 n2 * n3 /. An error condition results if the divisor is zero or if the quotient falls outside of the range {-32,768..32,767}.

*/MOD
"times-divide-mod"

STACK:

n1 n2 n3 -- rem quot

CATEGORY:

83 - Arithmetic

DESCRIPTION:

n1 is first multiplied by n2 producing an intermediate 32-bit result. rem is the remainder and quot is the floor of the quotient of the intermediate 32-bit result divided by the divisor n3. A 32-bit intermediate product is used as for */. rem has the same sign as n3 or is zero. An error condition results if the divisor is zero or if the quotient falls outside of the range {-32,768..32,767}.

*D
"times-d"

STACK:

n1 n2 --d

CATEGORY:

- Arithmetic

DESCRIPTION:

***D multiplies two single precision numbers n1 and n2 and leaves a double precision result d on the stack.**

+
"plus"

STACK:

w1 w2 -- w3

CATEGORY:

79 - Arithmetic

DESCRIPTION:

w3 is the arithmetic sum of w1 plus w2.

+!
"plus-store"

STACK:

w1 addr --

CATEGORY:

79 - Arithmetic

DESCRIPTION:

w1 is added to the w value at addr using the convention for +. This sum replaces the original value at addr.

+-
"plus-minus"

STACK:

w1 w2 -- w3

CATEGORY:

- Arithmetic

DESCRIPTION:

The +- operation negates the sign of w1 if the sign of w2 is negative. The top value w2 is then dropped.

+LOOP
"plus-loop"

STACK:

n --
sys -- (compiling)

CATEGORY:

C,I,83 - Control

DESCRIPTION:

n is added to the loop index. If the new index was incremented across the boundary between limit-1 and limit then the loop is terminated and loop control parameters are discarded. When the loop is not terminated, execution continues to just after the corresponding DO. sys is balanced with its corresponding DO. See: DO

, "comma"

STACK:

16b --

CATEGORY:

79 - Compiler word

DESCRIPTION:

ALLOT space for 16b then store 16b at HERE -2.

- "minus"

STACK:

w1 w2 -- w3

CATEGORY:

79 - Arithmetic

DESCRIPTION:

w3 is the result of subtracting w2 from w1.

-1 "minus-one"

STACK:

-- -1

CATEGORY:

- Constant, Arithmetic

DESCRIPTION:

-1 is a single-precision constant with the signed, single-precision value of -1. Since this value is used very often, it has been made into a constant in order to save compiling time.

-2 "minus-two"

STACK:

-- -2

CATEGORY:

- Constant, Arithmetic

DESCRIPTION:

-2 is a single-precision constant with the signed, single-precision value of -2. Since this value is used very often, it has been made into a constant in order to save compiling time.

-2ROT
"minus-two rote"

STACK:

32b1 32b2 32b3 --32b3 32b1 32b2

CATEGORY:

- Stack

DESCRIPTION:

The top three double-number stack entries are rotated, bringing the 2nd item to the top of the stack.

-3
"minus-three"

STACK:

-- -3

CATEGORY:

- Constant, Arithmetic

DESCRIPTION:

-3 is a single-precision constant with the signed, single-precision value of -3. Since this value is used very often, it has been made into a constant in order to save compiling time.

-3ROT
"minus-three rote"

STACK:

48b1 48b2 48b3 -- 48b3 48b1 48b2

CATEGORY:

- Stack

DESCRIPTION:

The top three triple-number stack entries are rotated, bringing the 2nd item to the top of the stack.

-4
"minus-four"

STACK:

--

CATEGORY:

- Constant, Arithmetic

DESCRIPTION;

-4 is a single-precision constant with the signed single-precision value of -4.

-ROT
"dash-rote"

STACK:

16b1 16b2 16b3 -- 16b3 16b1 16b2

CATEGORY:

- Stack

DESCRIPTION:

The top three stack entries are rotated, bringing the 2nd item to the top of the stack.

-TRAILING
"dash-trailing"

STACK:

addr +n1 -- addr +n2

CATEGORY:

79 - String

DESCRIPTION:

The character count +n1 of a text string beginning at addr is adjusted to exclude trailing spaces. If +n1 is zero, then +n2 is also zero. If the entire string consists of spaces, then +n2 is zero.

.
"dot"

STACK:

n --

CATEGORY:

79 - Output

DESCRIPTION:

The absolute value of n is displayed in a free-field format with a leading minus sign if n is negative.

."
"dot-quote"

STACK:

--
-- (compiling)

CATEGORY:

C,I,83 - Output

DESCRIPTION:

Used in the form: ." ccc"

Later execution will display the characters ccc up to, but not including, the delimiting " (close-quote). The blank following ." is not part of ccc.

.(
"dot-paren"

STACK:

--
-- (compiling)

CATEGORY:

I,83 - Output

DESCRIPTION:

Used in the form: .(ccc)

The characters ccc up to, but not including, the delimiting) (closing parenthesis) are displayed. The blank following .(is not part of ccc.

.NAME
"dot-name"

STACK:

addr -- Output

CATEGORY:

- System

DESCRIPTION:

.NAME is used to print the name-field of a dictionary header given the header's address (addr). The name is printed out the currently selected communications port.

.S
"dot-S"

STACK:

-- Output

CATEGORY:

- System

DESCRIPTION:

.S displays the contents of the stack.

/
"divide"

STACK:

n1 n2 -- n3

CATEGORY:

83 - Arithmetic

DESCRIPTION:

n3 is the floor of the quotient of n1 divided by n2. An error condition results if the divisor is zero or if the quotient is outside the range (-32,768 - 32,767).

/MOD
"divide-mod"

STACK:

n1 n2 -- rem quot

CATEGORY:

83 - Arithmetic

DESCRIPTION:

rem is the remainder and quot the floor of the quotient of n1 divided by the divisor n2. n3 has the same sign as n2 or is zero. An error condition results if the divisor is zero or if the quotient falls outside the range {-32,768..32,767}.

0
"zero"

STACK:

-- 0

CATEGORY:

- Arithmetic, Constant

DESCRIPTION:

0 is a single-precision constant with the signed, single-precision value of 0. Since this value is used very often, it has been made into a constant in order to save compiling time.

0<
"zero-less"

STACK:

n -- flag

CATEGORY:

83 - Comparison

DESCRIPTION:

flag is true if n is less than zero (negative).

0<>
"zero not-equal-to"

STACK:

n -- flag

CATEGORY:

- Comparison

DESCRIPTION:

flag will be true if n is not equal to zero.

0=
"zero-equals"

STACK:

w -- flag

CATEGORY:

83 - Comparison

DESCRIPTION:

flag is true if w is zero.

0>
"zero-greater"

STACK:

n -- flag

CATEGORY:

83 - Comparison

DESCRIPTION:

flag is true if n is greater than zero.

1
"one"

STACK:

-- 1

CATEGORY:

- Constant, Arithmetic

DESCRIPTION:

1 is a single-precision constant with the signed value of 1. Since this value is used very often, it has been made into a constant in order to save compiling time.

1+
"one-plus"

STACK:

w1 -- w2

CATEGORY:

79 - Arithmetic

DESCRIPTION:

w2 is the result of adding one to w1 according to the operation of + .

1-
"one-minus"

STACK:

w1 -- w2

CATEGORY:

79 - Arithmetic

DESCRIPTION:

w2 is the result of subtracting one from w1 according to the operation of - .

2
"two"

STACK:

-- 2

CATEGORY:

- Constant, Arithmetic

DESCRIPTION:

2 is a single-precision constant with the signed, single-precision value of 2. Since this value is used very often, it has been made into a constant in order to save compiling time.

2!
"two-store"

STACK:

d addr --

CATEGORY:

79 - Memory

DESCRIPTION:

2! stores a double-precision number d at the specified memory location addr.

2#IN
"two-number-in"

STACK:

-- d

CATEGORY:

- Input, Conversion

DESCRIPTION:

2#IN converts numeric characters received at the input port into a double- precision number and places this number on the stack. Conversion stops if a non-numeric character is received. If the number is larger than a double-precision value, the upper bits of the result will be incorrect.

2*
"two-times"

STACK:

n1 -- n2

CATEGORY:

79 - Arithmetic

DESCRIPTION:

n2 is the result of arithmetically shifting n1 left one bit. A zero is shifted into the vacated bit position.

2+
"two-plus"

STACK:

w1 -- w2

CATEGORY:

79 - Arithmetic

DESCRIPTION:

w2 is the result of adding two to w1 according to the operation of +.

2-
"two-minus"

STACK:

w1 -- w2

CATEGORY:

79 - Arithmetic

DESCRIPTION:

w2 is the result of subtracting two from w1 according to the operation of - .

2/
"two-divide"

STACK:

n1 -- n2

CATEGORY:

83 - Arithmetic

DESCRIPTION:

n2 is the result of arithmetically shifting n1 right one bit. The sign is included in the shift and remains unchanged.

2@
"two-fetch"

STACK:

addr -- 32b

CATEGORY:

79 - Memory

DESCRIPTION:

2@ will copy a double-precision number from the specified address to the top of the stack.

2ARRAY
"two-array"

STACK:

n --

CATEGORY:

- Defining word

DESCRIPTION:

2ARRAY is a defining word used to allocate memory for storage of a double-precision array. The format for using <name> is:

n <name>

n is the array element you want to access. When <name> is executed, the address of the specified array element is placed on the stack.

2CONSTANT
"two-constant"

STACK:

32b --

CATEGORY:

83 - Defining word

DESCRIPITON:

A defining word for creating a double precision constant used in the form:

d 2CONSTANT <name>

Creates a dictionary entry for <name> so that when <name> is later executed, the double-precision number d will be left on the stack.

2DROP
"two-drop"

STACK:

32b --

CATEGORY:

79 - Stack

DESCRIPITON:

2DROP will remove one double-precision number or two single-precision numbers from the top of the stack.

2DUP
"two-dupe"

STACK:

32b -- 32b 32b

CATEGORY:

79 - Stack

DESCRIPTION:

2DUP will duplicate the double-precision number on the top of the stack. It is a faster and more compact operation than OVER OVER.

2OVER
"two-over"

STACK:

32b1 32b2 -- 32b1 32b2 32b1

CATEGORY:

79 - Stack

DESCRIPTION:

2OVER copies the second double-precision number on the stack to the top of the stack.

2ROT
"two-rote"

STACK:

32b1 32b2 32b3 -- 32b2 32b3 32b1

CATEGORY:

- Stack

DESCRIPTION:

2ROT rotates the top three double-numbers on the stack, bringing the third double-number to the top of the stack.

2SWAP
"two-swap"

STACK:

32b1 32b2 -- 32b2 32b1

CATEGORY:

79 - Stack

DESCRIPTION:

2SWAP exchanges the top two double-precision numbers on the stack.

2VARIABLE
"two-variable"

STACK:

--

CATEGORY:

79 - Defining word

DESCRIPTION:

2VARIABLE is a defining word used to allocate memory for storage of a double-precision number. Used in the form:

2VARIABLE <name>

A dictionary entry for <name> **is created and four bytes are ALLOTted in its parameter field. The parameter field is to be used for contents of the variable. The application is responsible for initializing the contents of the variable which it creates. When** <name> **is later executed, the address of its parameter field is left on the stack.**

3
"three"

STACK:

-- 3

CATEGORY:

- Constant, Arithmetic

DESCRIPTION:

3 is a single-precision constant with the signed single-precision value of 3. Since this value is used very often, it has been made into a constant in order to save compiling time.

3DROP
"three-drop

STACK:

48b1 --

CATEGORY:

- Stack

DESCRIPTION:

3DROP will remove one triple-precision number from the top of the stack.

3DUP
"three-dupe"

STACK:

48b1-- 48b1 48b1

CATEGORY:

- Stack

DESCRIPTION:

3DUP will duplicate thetriple-precision number on the top of the stack.

3ROT
"three-rote"

STACK:

48b1 48b2 48b3 --48b2 48b3 48b1

CATEGORY:

- Stack

DESCRIPTION:

3ROT will rotate the top three triple-number, stack entries, bringing the 3rd item to the top of the stack.

3SWAP
"three-swap"

STACK:

48b1 48b2 -- 48b2 48b1

CATEGORY:

- Stack

DESCRIPTION:

3SWAP exchanges the top two triple-precision numbers on the stack.

4
"four"

STACK:

-- 4

CATEGORY:

- Constant, Arithmetic

DESCRIPTION:

4 is a single-precision constant with the signed single-precision value of 4.

:
"colon"

STACK:

-- sys

CATEGORY:

79 - Compiler word

DESCRIPTION:

A defining word executed in the form:

: <name> ... ;

Create a word definition for <name> in the compilation vocabulary and set compilation state. The search order is changed so that the first vocabulary in the search order is replaced by the compilation vocabulary. The compilation vocabulary is unchanged. The text from the input stream is subsequently compiled. <name> is called a "colon definition". The newly created word definition for <name> cannot be found in the dictionary until the corresponding ; is successfully processed.

An error condition exists if a word is not found and cannot be converted to a number or if, during compilation from mass storage, the input stream is exhausted before encountering ;. sys is balanced with its corresponding ; +.

;
"semi-colon"

STACK:

-- sys -- (compiling)

CATEGORY:

C,I,79 - Compiler word

DESCRIPTION:

Stops compilation of a colon definition, allows the <name> of this colon definition to be found in the dictionary, sets interpret state and compiles EXIT (or a system dependent word which performs an equivalent function). sys is balanced with its corresponding : .

<
"less-than"

STACK:

n1 n2 -- flag

CATEGORY:

83 - Comparison

DESCRIPTION:

flag is true if n1 is less than n2. -32768 < 32767 must return true. -32768 < 0 must return true.

<#
"less-sharp"

STACK:

--

CATEGORY:

79 - Conversion

DESCRIPTION:

Initialize pictured, numeric, output conversion. The words:

 # #> #S <# HOLD SIGN

can be used to specify the conversion of a double number into an ASCII text string stored in right-to-left order.

<>
"not-equal-to"

STACK:

n1 n2 -- f

CATEGORY:

- Comparison

DESCRIPTION:

Flag f is true if n1 is not equal to n2.

<MARK"
"from-mark"

STACK:

-- addr

CATEGORY:

- Control

DESCRIPTION:

<MARK places the value of the current dictionary pointer on the stack.

<RESOLVE
"from resolve"

STACK:

addr --

CATEGORY:

- Control

DESCRIPTION:

<RESOLVE compiles the word at addr into the current definition.

=
"equals"

STACK:

w1 w2 -- flag

CATEGORY:

- Comparison

DESCRIPTION:

flag is true if w1 is equal to w2

>
"greater-than"

STACK:

n1 n2 -- flag

CATEGORY:

83 - Comparison

DESCRIPTION:

flag is true if n1 is greater than n2. -32768 > 32767 must return false. -32768 > 0 must return false

>BODY
"to-body"

STACK:

addr1 -- addr2

CATEGORY:

83 - System

DESCRIPTION:

addr2 is the parameter-field address corresponding to the compilation address addr1.

>IN
"to-in"

STACK:

--addr

CATEGORY:

U,79 - System

DESCRIPTION:

>IN is the address of a variable which contains the present character offset within the input stream {{0..the number of characters in the input stream}}. See: WORD

>LINK
"to-link"

STACK:

addr1 -- addr2

CATEGORY:

- Compiler Word

DESCRIPTION:

addr2 is the link-field address corresponding to the compilation address addr1.

>MARK
"to-mark"

STACK:

-- addr

CATEGORY:

- Control

DESCRIPTION:

>MARK places the address of the next free location of a definition on the stack and places a temporary zero in that location. >RESOLVE is used later to replace the zero with the current dictionary pointer.

>NAME
"to-name"

STACK:

addr1 -- addr2

CATEGORY:

- Compiler word

DESCRIPTION:

addr2 is the name-field address corresponding to the compilation address addr1.

>R
"to-r"

STACK:

16b --

CATEGORY:

C,79 - Stack

DESCRIPTION:

Transfers 16b to the return stack.

>RESOLVE
"to-resolve"

STACK:

addr --

CATEGORY:

- Control

DESCRIPTION:

>RESOLVE places the current dictionary pointer at the address on the stack.

?
"query"

STACK:

addr --

CATEGORY:

-Output, Conversion

DESCRIPTION:

? performs a binary to ASCII conversion of the signed 16-bit contents of the specified memory location (addr) and prints it out the selected communications port.

?<MARK
"query-from-mark"

STACK:

-- flag addr

CATEGORY:

- Control

DESCRIPTION:

?<MARK will put a true flag on the stack and then execute <MARK.

?<RESOLVE
"query-from-resolve"

STACK:

flag addr --

CATEGORY:

- Control

DESCRIPTION:

?<RESOLVE will execute <RESOLVE if flag f is true. If flag f is false, ABORT" Conditionals Wrong" will be executed.

?>MARK
"query-to-mark"

STACK:

-- flag addr

CATEGORY:

- Control

DESCRIPTION:

?>MARK will put a true flag on the stack and then execute >MARK.

?>RESOLVE
"query-to-resolve"

STACK:

flag addr --

CATEGORY:

- Control

DESCRIPTION:

?>RESOLVE will execute >RESOLVE if flag f is true. If flag f is false, ABORT" Conditionals Wrong" will be executed.

?BRANCH
"query-branch"

STACK:

flag --

CATEGORY:

- Control

DESCRIPTION:

?BRANCH will execute BRANCH if flag f is zero.

?COMP
"query-comp"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

?COMP causes an error message to be issued if not compiling.

?CONDITION
"query-condition"

STACK:

flag --

CATEGORY:

- Comparison

DESCRIPTION:

?CONDITION will execute an ABORT" Conditionals Wrong" if flag f is false.

?CSP
"query-c-s-p"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

?CSP issues an error message and a QUIT if the current parameter-stack pointer position does not equal the value stored in the user variable CSP. Refer to WARNING and MESSAGE for information on the type of error message issued by ?CSP.

?DO
"query-do"

STACK:

n1 n2 --

CATEGORY:

C,I - Control

DESCRIPTION:

?DO is used to indicate the start of an iterative loop. ?DO is used along with LOOP or +LOOP within a colon-definition. n1 is the loop limit and n2 is the loop index. If the initial loop index n2 is equal to the limit n1, the loop is not executed. Instead, control will pass directly to the word following the LOOP or +LOOP word. Otherwise, the loop will terminate when the n2 is incremented past the boundary between n1-1 and n1.

?DUP
"query-dupe"

STACK:

16b -- 16b 16b or 0 -- 0

CATEGORY:

79 - Stack

DESCRIPTION:

?DUP will duplicate 16b if it is non-zero.

?EXEC
"query-exec"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

?EXEC causes an error message to be issued if not executing.

?KEY
"query-key"

STACK:

-- flag

CATEGORY:

- System

DESCRIPTION:

?KEY returns a flag indicating whether a character has appeared at the selected communications port. The flag is true if a character is present.

?LEAVE
"query-leave"

STACK:

flag --

CATEGORY:

- Control

DESCRIPTION:

?LEAVE will execute LEAVE if flag is true.

?PAIRS
"query-pairs"

STACK:

16b1 16b2 --

CATEGORY:

- System

DESCRIPTION:

?PAIRS issues an error message and executes an ABORT if the top 2 items on the stack don't match.

?STACK
"query-stack"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

?STACK checks if the stack pointer is out of bounds. If true, an error message is generated and program execution stops.

@
"fetch"

STACK:

addr -- 16b

CATEGORY:

- Memory

DESCRIPTION:

16b is the value at addr.

@@
"fetch-fetch"

STACK:

addr -- 16b

CATEGORY:

- Memory

DESCRIPTION:

@@ will copy a single-precision number from memory to the top of the stack. @@ is equivalent to doing @ @.

ABORT

STACK:

n... n --

CATEGORY:

79 - System

DESCRIPTION:

Abort clears the data stack and performs the function of QUIT. No message is displayed.

ABORT"
"abort-quote"

STACK:

flag -- -- (compiling)

CATEGORY:

C,I,83 - System

DESCRIPTION:

Used in the form:

flag ABORT" ccc"

When later executed, if flag is true the characters ccc, delimited by " (close-quote), are displayed and then a system dependent error abort sequence, including the function of ABORT, is performed. If flag is false, the flag is dropped and execution continues. The blank following ABORT" is not part of ccc.

ABS
"absolute"

STACK:

n -- u

CATEGORY:

79 - Arithmetic

DESCRIPTION:

u is the absolute value of n. If n is -32,768 then u is the same value.

ADDRESS

STACK:

-- addr

CATEGORY:

- Memory

DESCRIPTION:

ADDRESS points to "address" variable which is passed to-and-from the OPTOWARE™ driver. This word is used for storing or fetching a value in the "address" variable within the parameter block being pointed at by the "parameters" variable. The ADDRESS pointer is equal to the contents of PARAMETERS plus an offset of 2.

AGAIN

STACK:

--

CATEGORY:

C,I - Control

DESCRIPTION:

AGAIN causes program control to branch to the word immediately following the corresponding BEGIN.

ALLOT

STACK:

w --

CATEGORY:

79 - System

DESCRIPTION:

ALLOT allocates w bytes in the dictionary. The address of the next available dictionary location is updated accordingly.

AND

STACK:

16b1 16b2 -- 16b3

CATEGORY:

79 - Logical

DESCRIPTION:

16b3 is the bit-by-bit logical 'and' of 16b1 with 16b2.

ARRAY

STACK:

n --

CATEGORY:

- Defining word

DESCRIPTION:

ARRAY is a defining word used to allocate memory for storage of a single-precision array. Used in the form:

n ARRAY <name>

n is the array element you want to access. When <name> is executed, the address of the specified array element is placed on the stack.

ASCII
"as-key"

STACK:

--

sys --

CATEGORY:

- Compiling

DESCRIPTION:

Used in the form:

ASCII ccc

where the delimiter of ccc is a space. char is the ASCII character value of the first character in ccc. If interpreting, char is left on the stack. If compiling, compiles char as a literal so that when the colon definition is later executed, char is left on the stack.

AWAKE

STACK:

n --

--

CATEGORY:

Comparison, LC2/4

DESCRIPTION:

The AWAKE word is used to enable a single LC2/LC4 or all LC2/LC4's on a multidropped link which has been previously disabled with the SLEEP word. n represents the unique address of the LC2/LC4 to enable. When AWAKE is issued, each LC2/LC4 that is disabled will compare the value on the top of the stack to the value stored in its system variable LC.ADDRESS. If a match is found, that particular LC2/LC4 will be enabled and will respond to all communication on the host link. If no value appears on the stack, all LC2/LC4's will be enabled. See: SLEEP and LC.ADDRESS.

BANK.!
"bank-dot-store"

STACK:

16b addr --

CATEGORY:

- Memory

DESCRIPTION:

BANK.! stores 16b at addr in banked memory.

BANK.2!
"bank-dot-two-store"

STACK:

32b addr --

CATEGORY:

- Memory

DESCRIPTION:

BANK.2! stores a double-precision number, 32b, at the specified memory location addr in banked memory.

BANK.2@
"bank-dot-two-fetch"

STACK:

addr -- 32b

CATEGORY:

- Memory

DESCRIPTION:

32b is the value at addr in banked memory.

BANK.@
"bank-dot-fetch"

STACK:

addr -- 16b

CATEGORY:

- Memory

DESCRIPTION:

16b is the value at addr in banked memory.

BANK.C!
"bank-dot-c-store"

STACK:

16b addr --

CATEGORY:

- Memory

DESCRIPTION:

The least-significant 8-bits of 16b are stored into the byte at addr in banked memory.

BANK.C@
"bank-dot-c-fetch"

STACK:

addr -- 16b

CATEGORY:

- Memory

DESCRIPTION:

BANK.C@ returns the least significant byte at addr in banked memory. The most significant byte is zero.

BASE

STACK:

-- addr

CATEGORY:

U,83 - System

DESCRIPTION:

The address of a variable containing the current numeric conversion radix. {{2..72}}

BEGIN

STACK:

--

CATEGORY:

C,I,79 - Control

DESCRIPTION:

Used in the form:

BEGIN ... flag UNTIL
or
BEGIN ... flag WHILE ... REPEAT

BEGIN marks the start of a word sequence for repetitive execution. A BEGIN-UNTIL loop will be repeated until flag is true. A BEGIN-WHILE-REPEAT loop will be repeated until flag is false. The words after UNTIL or REPEAT will be executed when either loop is finished. sys is balanced with its corresponding UNTIL or WHILE.

BL
"b-l"

STACK:

-- 32 (decimal)
-- 20 (hex)

CATEGORY:

- String

DESCRIPTION:

BL is a single precision constant which places the ASCII value for the "space" character on the stack. This value is a 20 in hex or a 32 in decimal.

BODY>
"body-to"

STACK:

addr1 -- addr2

CATEGORY:

- Conversion

DESCRIPTION:

BODY> converts the parameter field address addr1 to the compilation address addr2.

BRANCH

STACK:

--

CATEGORY

- Compiler word

DESCRIPTION:

BRANCH is a run-time procedure to unconditionally branch. The interpretive pointer IP is replaced by the value following the BRANCH instruction.

C!
"c-store"

STACK:

16b addr --

CATEGORY:

79 - Memory

DESCRIPTION:

The least-significant 8 bits of 16b are stored into the byte at addr.

C,
"c-comma"

STACK:

16b --

CATEGORY:

79 - Compiler word

DESCRIPTION:

C, will store an 8-bit byte directly into the FORTH dictionary. The low order 8 bits of the word on the top of the stack are stored at HERE (the next available dictionary location). The dictionary pointer is then increased by 1. This word is equivalent to: HERE C! 1 ALLOT

C/L
"character-per-line"

STACK:

--- n

CATEGORY:

- System (64)

DESCRIPTION:

C/L is a constant leaving the number of characters per line.

CASE

STACK:

--

CATEGORY:

- Control

DESCRIPTION:

Start of CASE control structure.

C@
"c-fetch"

STACK:

addr -- 16b

CATEGORY:

79 - Memory

DESCRIPTION:

C@ returns the least-significant byte at addr. The most-significant byte is zero.

CLEAR.BUF
"clear-dot-buf"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

CLEAR.BUF resets the buffer pointers of the currently selected communications port.

CLOCK.TICK
"clock-dot-tick"

STACK:

-- addr

CATEGORY:

- System, LC2/4

DESCRIPTION:

CLOCK.TICK points to a variable which contains a value that is updated by the local controller's internal timer. On an LC2, this variable points to a 16-bit number which is incremented every one-tenth of a second. On an LC4, this variable points to a 32-bit value incremented every one- hundredth of a second. This is useful for executing words at precise times.

CMOVE
"c-move"

STACK:

addr1 addr2 u --

CATEGORY:

83 - Memory

DESCRIPTION:

Move u bytes beginning at address addr1 to addr2. The byte at addr1 is moved first, proceeding toward high memory. If u is zero, nothing is moved.

CMOVE>
"c-move-up"

STACK:

addr1 addr2 u --

CATEGORY:

83 - Memory

DESCRIPTION:

CMOVE moves the u bytes at address addr1 to addr2. The move begins by moving the byte at (addr1 plus u minus 1) to (addr2 plus u minus 1) and proceeds to successively lower addresses for u bytes. If u is zero, nothing is moved. (Useful for sliding a string towards higher addresses).

COLD

STACK:

--

CATEGORY:

- System

DESCRIPTION:

COLD performs a "cold-start" in LC2/LC4 FORTH by first initializing the user variables to their start-up values then calling the word ABORT.

COM1
"com-one"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

COM1 selects LC2/LC4's OPTOMUX™ communications port as the current input/output device. The word CONSOLE can be used to switch back to the host port. Using COM1 is not necessary when OPTOWARE™ is called, because OPTOWARE™ addresses the OPTOMUX™ port directly.

COM2 (LC4)
"com-two"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

COM2 selects communications port 2 located on the EX2 daughter card as the current input/output device. The EX2 daughter card plugs onto the expansion port connector or an LC4. The word console can be used to switch back to the host port.

This word is available only with LC4.

COM3 (LC4)
"com-three"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

COM3 selects communications port3 located on the EX2 daughter card as the current input/output device. The EX2 daughter card plugs onto the expansion port connector of an LC4. The word console can be used to switch back to the host port.

This word is available only with LC4.

COMMAND

STACK:

-- addr

CATEGORY:

- System, LC2/4

DESCRIPTION:

COMMAND points to "command" variable which is passed to and from the OPTOWARE™ driver. This word is used for storing or fetching a value in the "command" variable within the parameter block being pointed at by the "parameters" variable. The COMMAND pointer is equal to the contents of PARAMETERS plus an offset of 4.

COMPILE

STACK:

--

CATEGORY:

C,83 - Compiler word

DESCRIPTION:

Typically used in the form:

: <name>... COMPILE <namex> ...;

When <name> is executed, the compilation address compiled for <namex> is compiled and not executed. <name> is typically immediate and <namex> is typically not immediate.

CONSOLE

STACK:

--

CATEGORY:

- System

DESCRIPTION:

Selects LC2/LC4's host communications port as the current output device.

CONSTANT

STACK:

16b --

CATEGORY:

83 - Defining word

DESCRIPTION:

CONSTANT is a defining word executed in the form:

16b CONSTANT <name>

CONSTANT creates a dictionary entry for <name> so that when <name> is later executed, 16b will be left on the stack.

CONTEXT

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

CONTEXT is a user variable which contains a pointer to the vocabulary which is to be searched first. CONTEXT is set to point to a specific vocabulary by executing that vocabulary name.

CONVERT

STACK:

+d1 addr1 -- +d2 addr2

CATEGORY:

79 - Conversion

DESCRIPTION:

+d2 is the result of converting the characters within the text beginning at addr1+1 into digits, using the value of BASE, and accumulating each into +d1 after multiplying +d1 by the value of BASE. Conversion continues until an unconvertible character is encountered. addr2 is the location of the first unconvertible character.

COUNT

STACK:

addr1 -- addr2 +n

CATEGORY:

79 - String

DESCRIPTION:

addr2 is addr1+1 and +n is the length of the counted string at addr1. The byte at addr1 contains the byte count +n. Range of +n is {0..255}.

CR
"c-r"

STACK:

--

CATEGORY:

79 - Output

DESCRIPTION:

CR displays a carriage-return and line-feed or equivalent operation.

CREATE

STACK:

--

CATEGORY:

79 - Compiler word

DESCRIPTION:

CREATE is a defining word executed in the form: CREATE <name> Creates a dictionary entry for <name>. After <name> is created, the next available dictionary location is the first byte of <name>'s parameter field. When <name> is subsequently executed, the address of the first byte of <name>'s parameter field is left on the stack. CREATE does not allocate space in <name>'s parameter field.

CSP
"c-s-p"

STACK:

-- addr

CATEGORY:

U - System

DESCRIPTION:

CSP is a user variable which is used as a temporary location for the compiler stack pointer position. addr is the address of the user variable CSP.

CURRENT

STACK:

-- addr

CATEGORY:

U - System

DESCRIPTION:

CURRENT is a user variable that contains a pointer to the vocabulary to which definitions are "currently" being appended to. CURRENT is set to point to a specific vocabulary by executing the word DEFINITIONS, that copies the vocabulary pointer in CONTEXT into CURRENT.

D+
"d-plus"

STACK:

wd1 wd2 --wnd3

CATEGORY:

79 - Arithmetic

DESCRIPTION:

D+ is used to add two double-precision numbers. wd3 is the arithmetic sum of wd1 plus wd2.

D-
"d-minus"

STACK:

wd1 wd2 -- wd3

CATEGORY:

79 - Arithmetic

DESCRIPTION:

D- is used to subtract two double-precision numbers. wd3 is the result of subtracting wd2 from wd1.

D+-
"D-plus-minus"

STACK:

d1 n -- d2

CATEGORY:

- Arithmetic

DESCRIPTION:

The sign of the double-precision value d1 is negated if the sign of the single- precision value n is negative. Then the value n is dropped from the top of the stack.

D.
"d-dot"

STACK:

d --

CATEGORY:

- Output

DESCRIPTION:

D. prints a signed double-precision number followed by a trailing blank.

D.R
"d-dot-r"

STACK:

d width --

CATEGORY:

- Output

DESCRIPTION:

D.R prints a signed double-precision number, right-justified in a field of specified width. If the field width is smaller than the number of significant digits, the number is printed without leading blanks. No trailing blanks are printed.

D0=
"d-zero-equal"

STACK:

wd -- flag

CATEGORY:

- Comparison

DESCRIPTION:

D0= compares a double-precision number wd on the top of the stack with zero. If the number d is equal to zero, a true flag is left on the stack.

D2*
"d-two-times"

STACK:

d1 -- d2

CATEGORY:

- Arithmetic

DESCRIPTION:

d2 is the result of arithmetically shifting d1 left one bit. A zero is shifted into the vacated bit position.

D<
"d-less-than"

STACK:

d1 d2 -- flag

CATEGORY:

83 - Comparison

DESCRIPTION:

D< compares two signed, double-precision numbers d1 and d2. If d1 is less than the number on the top of the stack d2, a true flag is left on the stack.

D=
"D-equals"

STACK:

wd1 wd2 -- flag

CATEGORY:

- Comparison

DESCRIPTION:

D= compares two double-precision numbers wd1 and wd2. If d1 and d2 are equal, a true flag is left on the stack.

D>
"D-greater-than"

STACK:

d1 d2 -- flag

CATEGORY:

- Comparison

DESCRIPTION:

D> compares two signed, double-precision numbers d1 and d2. If d1 is greater than the number on the top of the stack d2, a true flag is left on the stack.

D>F
"d-to-f"

STACK:

d -- float

CATEGORY:

- Conversion

DESCRIPTION:

D>F converts the double-precision integer on the stack into a floating-point number.

DABS
"d-abs"

STACK:

d -- ud

CATEGORY:

79 - Arithmetic

DESCRIPTION:

ud is the absolute value of d. If d is -2,147,483,648 then ud is equal to d.

DATE!
"date-store"

STACK:

n1 n2 n3 --

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

DATE! is used to set the real-time clock's date. n1 is a number representing the month (1-12), n2 is the day (1-31), and n3 represents the year (0-99).

DATE@
"date-fetch"

STACK:

-- n1 n2 n3

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

DATE@ is used to read the real-time clock's date. n1 is a number representing the month (1-12), n2 is the day (1-31), and n3 represents the year (0-99).

DAYS!
"days-store"

STACK:

n --

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

DAYS! is used to set the day on the real-time clock. n is in the range of 1 to 31.

DAYS@
"days-fetch"

STACK:

-- n

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

DAYS@ is used to read the day on the real-time clock. n is in the range of 1 to 31.

DECIMAL

STACK:

--

CATEGORY:

79 - Conversion

DESCRIPTION:

DECIMAL sets the input-output numeric conversion base to ten.

DEFINITIONS

STACK:

--

CATEGORY:

79 - System

DESCRIPTION:

The compilation vocabulary is changed to be the same as the first vocabulary in the search order.

DEPTH

STACK:

-- +n

CATEGORY:

79 - Stack

DESCRIPTION:

+n is the number of 16-bit values contained in the data stack before +n was placed on the stack.

DIGIT

STACK:

char n -- n1 flag (valid digit)
char n -- flag (invalid digit)

CATEGORY:

- Conversion

DESCRIPTION:

DIGIT converts a character char using base n to its binary equivalent n2. If the character is a legal digit in the specified radix n, then the digit's binary equivalent and a true flag are returned on the stack. If the character is an invalid digit, only a false flag is left on the stack.

DLITERAL
"d-literal"

STACK:

-- 32b (executing)
32b -- (compiling)

CATEGORY:

- Compiler word

DESCRIPTION:

DLITERAL will compile a double number into a literal definition. When executing the definition, the 32-bit number previously compiled will be pushed onto the stack.

DMAX
"d-max"

STACK:

d1 d2 -- d3

CATEGORY:

79 - Arithmetic

DESCRIPTION:

d3 is the larger of the two, signed, double-precision values d2 and d1.

DMIN
"d-min"

STACK:

d1 d2 -- d3

CATEGORY:

79 - Arithmetic

DESCRIPTION:

d3 is the smaller of the two, signed, double-precision values d2 and d1.

DNEGATE
"d-negate"

STACK:

d1 -- d2

CATEGORY:

79 - Arithmetic

DESCRIPTION:

d2 is the two's complement of d1.

DO

STACK:

w1 w2 --
-- sys (compiling)

CATEGORY:

C,I,83 - Control

DESCRIPTION:

Used in the form:

DO ... LOOP or DO ... +LOOP

Begins a loop which terminates based on control parameters. The loop index begins at w2, and terminates based on the limit w1. See LOOP and +LOOP for details on how the loop is terminated. The loop is always executed at least once. For example: w DUP DO ... LOOP executes 65,536 times. sys is balanced with its corresponding LOOP or +LOOP.

An error condition exists if insufficient space is available for at least three nesting levels.

DOES>
"does"

STACK:

-- addr
-- (compiling)

CATEGORY:

C,I,83 - Compiler word

DESCRIPTION:

Defines the execution-time action of a word created by a high-level defining word. Used in the form:

: <namex> ... <create> ... DOES> ... ;

and then

<namex> <name>

where <create> is CREATE or any user defined word which executes CREATE.

Marks the termination of the defining part of the defining word <namex> and then begins the definition of the execution-time action for words that will later be defined by <namex>. When <name> is later executed, the address of <name>'s parameter field is placed on the stack and then the sequence of words between DOES> and ; are executed.

DP
"d-p"

STACK:

-- addr

CATEGORY:

U - System

DESCRIPTION:

DP is a user variable that contains an address pointer of the next available dictionary location. addr is the address of the variable DP. It is used in conjunction with the word HERE. When HERE is executed, the 16-bit single precision contents of DP are placed on the stack. The words FORGET and ALLOT alter the contents of DP.

DPL
"d-p-l"

STACK:

-- addr

CATEGORY:

U - System

DESCRIPTION:

DPL is a user variable which indicates the number of digits found to the right of a decimal point when converting a numeric character string into a numeric value. addr is the address of the variable DPL. The word INTERPRET uses the contents of DPL to determine if a numeric value is to be treated as single- precision (DPL=-1, no decimal point) or double-precision (DPL not equal to -1, decimal point encountered).

DROP

STACK:

16b --

CATEGORY:

79 - Stack

DESCRIPTION:

16b is removed from the stack

DSHIFT
"d-shift"

STACK:

wd1 n -- wd2

CATEGORY:

- Arithmetic

DESCRIPTION;

wd2 is the result of shifting n-1 n bits. If n is negative, wd1 is shifted to the right; if n is positive, wd1 is shifted to the left.

DU.
"d-u-dot"

STACK:

ud --

CATEGORY:

- Output

DESCRIPTION;

d is displayed as an unsigned, double-number in a free-field format.

DU.R
"d-u-dot-r"

STACK:

ud n --

CATEGORY:

- Output

DESCRIPTION;

DU.R prints an unsigned, double number right-justified in a field of specified width. If the specified width is less than the number of significant digits, the number is printed without leading blanks. No trailing blanks are printed.

DU<
"d-u-less"

STACK:

ud1 ud2 -- flag

CATEGORY:

- Comparison

DESCRIPTION:

DU< compares two, unsigned double-numbers ud1 and ud2. A true flag is left on the stack if ud1 is less than the value on the top of the stack ud2.

DU>
"d-u-more"

STACK:

ud1 ud2 -- flag

CATEGORY

- Comparison

DESCRIPTION:

DU> does an unsigned comparison of ud1 and ud2. If d1 is greater than ud2, a true flag is returned. Otherwise the flag returned is false.

DUMP

STACK:

addr n --

CATEGORY:

- Output

DESCRIPTION;

Does a combined hex and ASCII dump of the specified memory. addr is address to start dumping from and n is the number of bytes to dump.

DUP
"dupe"

STACK:

16b -- 16b 16b

CATEGORY:

79 - Stack

DESCRIPTION:

16b is duplicated on the top of the stack.

E

STACK:

d -- float

CATEGORY:

- Conversion, Input

DESCRIPTION;

Used in the form:

d E n

E is used for entering a number in floating point format. d is a double- precision number used as the mantissa and n is a 16-bit integer used as the exponent.

ELSE

STACK:

--

sys1 -- sys2 (compiling)

CATEGORY:

C,I,79 - Control

DESCRIPTION:

Used in the form:

flag IF ... ELSE ... THEN

ELSE executes after the true part following IF. ELSE forces execution to continue at just after THEN. sys1 is balanced with its corresponding IF. sys2 is balanced with its corresponding THEN . See: IF THEN.

EMIT

STACK:

16b --

CATEGORY:

83 - Output

DESCRIPTION:

The least-significant 8-bit ASCII character is sent to the selected output port.

EMPTY

STACK:

--

CATEGORY:

- System

DESCRIPTION;

EMPTY does a FORGET of all the user compiled FORTH words.

ENCLOSE

STACK:

addr1 char -- addr1 n1 n2 n3

CATEGORY:

- String

DESCRIPTION:

ENCLOSE is a text parsing primitive used by WORD. addr1 is the begin- ning address of the text string to parse and char is an ASCII delimiting character. n1 is the offset to the first non-delimiter character. n2 is the offset to the first delimiter after the text. n3 is the offset to the first character used to start the next scan. This procedure will process past an ASCII null. The ASCII null will be treated as an unconditional delimiter.

ENDCASE

STACK:

addr1 addr2 ... addr n -- (compiling)
n -- (if no case was found)
-- (if case was found)

CATEGORY:

- Control

DESCRIPTION:

If no case was found, the case value is dropped. If the case was found, nothing is done.

ENDOF

STACK:

addr1 n1 -- addr2 n2

CATEGORY:

- Compiling

DESCRIPTION:

At run time, ENDOF transfers control to the code following the next ENDCASE provided there was a match at the last OF. If there was not a match at the last OF, ENDOF is the location to which execution branched.

At compile time, ENDOF compiles a BRANCH, reserves a branch address, and leaves the parameters addr2 and n2 on the stack. ENDOF also resolves the pending forward ?BRANCH from OF by calculating the offset from addr1 to HERE and storing it at addr1.

ERROR

STACK:

addr1 n1 -- addr2 n2

CATEGORY:

- System, LC2/4

DESCRIPTION:

ERROR prints a system error.

ERRORS

STACK:

-- addr

CATAGORY:

- Memory

DESCRIPTION:

ERRORS points to "errors" variable which is passed to and from the OPTOWARE™ driver. This word is used for storing or fetching a value in the "errors" variable within the parameter block being pointed at by the "parameters" variable. The ERRORS pointer is equal to the contents of parameters plus an offset of 0.

EVEN

STACK:

-- n

CATEGORY:

- Constant, System

DESCRIPTION:

A constant for the SET.SERIAL word for even parity.

EXECUTE

STACK:

addr --

CATEGORY:

79 - Control

DESCRIPTION:

The word definition indicated by addr is executed. An error condition exists if addr is not a compilation address.

EXIT

STACK:

--

CATEGORY:

C,79 - Control

DESCRIPTION:

When compiled within a colon definition, EXIT terminates execution of the definition at that point. . An error condition exists if the top of the return stack does not contain a valid return point. May not be used within a do- loop.

EXPECT

STACK:

addr +n --

CATEGORY:

83 - Input

DESCRIPTION:

Receive characters and store each into memory. The transfer begins at addr proceeding towards higher addresses one byte per character until either a "return" is received or until +n characters has been transferred. No more than +n characters will be stored. The "return" is not stored into memory. No characters are received or transferred if +n is zero. All characters actually received and stored into memory will be displayed, with the "return" displaying as a space.

F!
"f-store"

STACK:

float addr --

CATEGORY:

- Memory

DESCRIPTION:

F! will copy a floating point number from the second position on the stack to the address specified on top of the stack.

F*
"f-times"

STACK:

float1 float2 -- float3

CATEGORY:

- Arithmetic

DESCRIPTION:

F* multiplies float1 by float2 and returns float3.

F+
"f-plus"

STACK:

float1 float2 -- float3

CATEGORY:

- Arithmetic

DESCRIPTION:

float3 is the result of adding float1 and float2.

F-
"f-minus"

STACK:

float1 float2 -- float3

CATEGORY:

- Arithmetic

DESCRIPTION:

float3 is the result of subtracting float2 from float1.

F.
"f-dot"

STACK:

float --

CATEGORY:

- Output

DESCRIPTION:

The floating point number float will be displayed in a free-field format with a leading minus sign if float is negative.

F.R
"f-dot-r"

STACK:

float n --

CATEGORY:

- Output

DESCRIPTION:

F.R. prints a floating point number, right-justified in a field of specified width. If the field width is smaller than the number of significant digits, the number is printed without leading blanks. No trailing blanks are printed.

F/
"f-divide"

STACK:

float1 float2 -- float3

CATEGORY:

- Arithmetic

DESCRIPTION:

F/ divides float1 by float2 and returns float3.

F0=
"f-zero-equals"

STACK:

float -- flag

CATEGORY:

- Comparison

DESCRIPTION:

F0= checks for float to be equal to 0. If the float is not equal to 0, flag will be a false; else it will be a true.

F<
"f-less"

STACK:

float1 float2 -- flag

CATEGORY:

- Comparison

DESCRIPTION:

F< checks for float1 to be less than float2. If float1 is smaller, flag will be a true; else it will be a false.

F=
"f-equal

STACK:

float1 float2 -- flag

CATEGORY:

- Comparison

DESCRIPTION:

F= checks for float1 to be equal to float2. If the numbers are not equal, flag will be a false, else it will be a true.

F>
"f-more"

STACK:

float1 float2 -- flag

CATEGORY:

- Comparison

DESCRIPTION:

F> checks for float1 to be greater than float2. If float1 is greater, flag will be a true, else it will be a false.

F>D
"f-to-d"

STACK:

float -- d

CATEGORY:

- Conversion

DESCRIPTION:

F>D converts a floating-point number to a 32-bit integer number.

F?
"f-query"

STACK:

addr --

CATEGORY:

- Output

DESCRIPTION:

F? prints floating-point number stored at addr.

F@
"f-fetch"

STACK:

addr -- float

CATEGORY:

- Memory

DESCRIPTION:

F@ will copy a floating-point number from the specified address to the top of the stack.

FALSE

STACK:

-- 0

CATEGORY:

- Logical

DESCRIPTION:

FALSE is a constant which is equal to the value zero. It is useful for setting flags or logical states to a "false" value.

FARRAY
"f-array"

STACK:

n --

CATEGORY:

- Defining word

DESCRIPTION:

A defining word executed in the form:

n FARRAY <name>

A dictionary entry for <name> is created and n * 4 bytes are allotted in its parameter field. This parameter field is to be used for contents of the array. The application is responsible for initializing the contents of the array which it creates. The format for using <name> is:

n <name>

n is the array element you want to access. When <name> is executed, the address of the specified array element is placed on the stack.

FCONSTANT
"f-constant"

STACK:

float --

CATEGORY:

- Defining word

DESCRIPTION:

A defining word executed in the form:

loat FCONSTANT <name>

Creates a dictionary entry for <name> so that when <name> is later executed, float will be left on the stack.

FCOS
"f-cosine"

STACK:

float1 -- float2

CATEGORY:

- Conversion

DESCRIPTION:

FCOS returns the cosine of float1 in radians (float2).

FENCE

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

FENCE is a user variable which sets a boundary past which FORGET cannot forget.

FEXP
"f-exponent"

STACK:

float1 --float2

CATEGORY:

- Conversion

DESCRIPTION:

FEXP calculates the value of "e" (base value of natural logarithm = 2.71828) raised to the power float1.

FILL

STACK:

addr u 8b --

CATEGORY:

83 - Memory

DESCRIPTION:

u bytes of memory beginning at addr are set to 8b. No action is taken if u is zero.

FIND

STACK:

addr1 -- addr2 n

CATEGORY:

83 - System

DESCRIPTION:

addr1 is the address of a counted string. The string contains a word name to be located in the currently active search order. If the word is not found, addr2 is the string address addr1, and n is zero. If the word is found, addr2 is the compilation address and n is set to one of two non-zero values. If the word found has the immediate attribute, n is set to one. If the word is non-immediate, n is set to minus one (true).

FLN
"f-l-n"

STACK:

float1 -- float2

CATEGORY:

- Conversion

DESCRIPTION:

FLN returns the natural log of float1 (float2). If float1 is less than or equal to zero, FPERR will contain an error code.

FNEGATE
"f-negate"

STACK:

float1 -- float2

CATEGORY:

- Arithmetic

DESCRIPTION:

Float 2 is the negative of float1.

FORGET

STACK:

--

CATEGORY:

83 - System

DESCRIPTION:

Used in the form:

FORGET <name>

If <name> is found in the compilation vocabulary, delete <name> from the dictionary and all words added to the dictionary after <name> regardless of their vocabulary. Failure to find <name> is an error condition. An error condition also exists if the compilation vocabulary is deleted.

FORTH

STACK:

--

CATEGORY:

83 - System

DESCRIPTION:

The name of the primary vocabulary. Execution replaces the first vocabulary in the search order with FORTH. FORTH is initially the compilation vocabulary and the first vocabulary in the search order. New definitions become part of the FORTH vocabulary until a different compilation vocabulary is established. See: VOCABULARY

FORTH-83

STACK:

--

CATEGORY:

83 - System

DESCRIPTION:

Assures that a FORTH-83 Standard System is available, otherwise an error condition exists.

FPERR
"floating-point-error"

STACK:

-- addr

CATEGORY:

- Arithmetic

DESCRIPTION:

FPERR is the error variable used by the floating point routines. After each floating point operation, FPERR is set.

FPERR	Condition
0	no error
1	underflow
2	overflow
3	not a number

FSIN
"f-sine"

STACK:

float1 -- float2

CATEGORY:

- Arithmetic

DESCRIPTION:

FSIN returns the sine of float1 in radians (float2).

FSQRT
"f-square root"

STACK:

float1 -- float2

CATEGORY:

- Arithmetic

DESCRIPTION:

FSQRT returns the square root of float1 (float2). If float1 is negative, FPERR will contain an error code.

FVARIABLE

STACK:

--

CATEGORY:

- Defining word

DESCRIPTION:

A defining word executed in the form:

FVARIABLE <name>

A dictionary entry for <name> is created and four bytes are allotted in its parameter field. This parameter field is to be used for contents of the variable. The application is responsible for initializing the contents of the variable which it creates. When <name> is later executed, the address of its parameter field is placed n the stack.

F^
"f-power"

STACK:

float1 float2 -- float3

CATEGORY:

Arithmetic

DESCRIPTION:

F^ raises the number float1 to the float2 power, returning result float3.

HERE

STACK:

-- addr

CATEGORY:

79 - System

DESCRIPTION:

The address of the next available dictionary location.

HEX

STACK:

--

CATEGORY:

79 - Conversion

DESCRIPTION:

The system radix is set to 16. Used for inputting hex numbers or converting from an alternate base. See also: DECIMAL.

HLD
"h-l-d"

STACK:

-- addr

CATEGORY:

U - Conversion

DESCRIPTION:

HLD is a user variable that holds the address of the latest character of text during numeric output conversion.

HOLD

STACK:

char --

CATEGORY:

79 - Conversion

DESCRIPTION:

char is inserted into a pictured numeric output string. Typically used between <# and #>.

HOURS!
"hours-store"

STACK:

n --

CATEGORY

- Input/Output, LC2/4

DESCRIPTION:

HOURS! is used to set the hour on the real-time clock. n is in the range 0 to 23.

HOURS@
"hours-fetch"

STACK:

-- n

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

HOURS@ is used to read the hour on the real-time clock. n is in the range 0 to 23.

I

STACK:

-- w

CATEGORY:

C,79 - System

DESCRIPTION:

w is a copy of the loop index. May only be used in the form:

```
DO ... I ... LOOP
    or
DO ... I ... +LOOP
```

IF

STACK:

flag --
-- sys (compiling)

CATEGORY:

C,I,79 - Control

DESCRIPTION:

Used in the form:

flag IF ... THEN or IF...ELSE...THEN

If flag is true, the words following IF are executed and the words following ELSE until just after THEN are skipped. The ELSE part is optional.

If flag is false, words from IF through ELSE, or from IF through THEN (when no ELSE is used), are skipped. sys is balanced with its corresponding ELSE or THEN.

IMMEDIATE

STACK:

--

CATEGORY:

79 - System

DESCRIPTION:

Marks the most recently created dictionary entry as a word which will be executed when encountered during compilation rather than compiled.

INFO

STACK:

n -- addr

CATEGORY:

- Memory, LC2/4

DESCRIPTION:

Points to "info" array which is passed to-and-from the OPTOWARE™ driver. Used for storing or fetching a value in the "info" variable array within the parameter block being pointed at by the "parameters" variable. The info array is a 16-element array. n (range of 0 - 15), determines which element is to be accessed. Resulting INFO pointer is equal to the following FORTH expression:

PARAMETERS @ 42 + 2 n * +

INTERPRET

STACK:

--

CATEGORY:

- System

DESCRIPTION:

INTERPRET is the outer text interpreter which executes or compiles text sequentially from the input stream depending on STATE> If a word name cannot be found after searching the CONTEXT and the CURRENT vocab- ularies, it is converted to a number according to the current base. If the text cannot be converted to a number, an error message will occur which will print the name followed by a question mark (?). Text input is accepted according to the convention for WORD. If a decimal point is found as part of a number, that number will be treated as a double- number value.

J

STACK:

-- w

CATEGORY:

C,79 - System

DESCRIPTION:

w is a copy of the index of the next outer loop. May only be used within a nested DO-LOOP or DO-+LOOP in the form, for example:

DO ... DO ... J ... LOOP ... +LOOP

K

STACK:

-- w

CATEGORY:

C - System

DESCRIPTION:

w is a copy of the index of the next outer loop. May only be used within a nested DO-LOOP or DO-+LOOP in the form, for example:

DO ... DO ... DO ... K ... J ... I ... LOOP ... LOOP ... LOOP

KEY

STACK:

-- 16b

CATEGORY:

83 - Input

DESCRIPTION:

The least-significant 8 bits of 16b is the next ASCII character received from the selected communications port buffer. All valid ASCII characters can be received. Control characters are not processed by the system for any editing purpose. Characters received by KEY will be removed from the buffer and will not be echoed. Use of CLEAR.BUF is recommended to clear all previous characters in the buffer before KEY is used.

LATEST

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

LATEST places the name field address of the top-most word in the vocabulary on the stack. The topmost word is the most recently compiled definition.

LC.ADDRESS
"l-c-dot-address"

STACK:

addr --

CATEGORY:

- System, LC2/4

DESCRIPTION

LC.ADDRESS is a variable which contains the LC2 address for use with the SLEEP and AWAKE words. Assigning LC2 a unique address is only necessary when several LC2's are connected to the same communi- cations link. The LC.ADDRESS word should be issued to only one LC2 unit at a time. All other LC2 units should be powered down when an address is being set

The LC.ADDRESS variable is a special system variable which can only be cleared by reassigning it. It is not affected by power down conditions.

LEAVE

STACK:

--

-- **(compiling)**

CATEGORY:

C,I,83 - Control

DESCRIPTION:

Transfers execution to just beyond the next LOOP or +LOOP. The loop is ter- minated and loop control parameters are discarded. May only be used in the form:

DO ... LEAVE ... LOOP
or
DO ... LEAVE ... +LOOP

LEAVE may appear within other control structures which are nested within the do-loop structure. More than one LEAVE may appear within a do-loop.

LITERAL

STACK:

-- 16b 16b -- (compiling)

CATEGORY:

C,I,79 - System

DESCRIPTION:

Typically used in the form:

[16b] LITERAL

Compiles a system-dependent operation so that when later executed, 16b will be left on the stack.

LOC

STACK:

-- n

CATEGORY:

- Output

DESCRIPTION

LOC returns the number of bytes in the current serial port interrupt buffer.

LOF

STACK:

-- n

CATEGORY:

- Output

DESCRIPTION:

LOF returns the number of free spaces in the current serial port interrupt buffer. This is equivalent to 255 minus the value returned by LOC.

LOOP

STACK:

--
sys -- (compiling)

CATEGORY:

C,I,83 - Control

DESCRIPTION:

LOOP increments the DO-LOOP index by one. If the new index was incremented across the boundary between limit-1 and limit, the loop is terminated and loop control parameters are discarded. When the loop is not terminated, execution continues to just after the corresponding DO. sys is balanced with its corresponding DO. See: DO.

M/MOD
"M-divide-mod"

STACK:

d n -- rem quot

CATEGORY:

- Arithmetic

DESCRIPTION:

M/MOD is used to divide a double-precision number (d) by a single- precision number (n) and leave the remainder (rem) and quotient (quot) on the stack. The remainder is signed with the sign of the divisor and the quotient is floored.

MAX
"max"

STACK:

n1 n2 -- n3

CATEGORY:

79 - Arithmetic

DESCRIPTION:

n3 is the greater of n1 and n2 according to the operation of >.

MIN
"min"

STACK:

n1 n2 -- n3

CATEGORY:

79 - Arithmetic

DESCRIPTION:

n3 is the lesser of n1 and n2 according to the operation of <.

MINUTES!
"minutes-store"

STACK:

n --

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

MINUTES! is used to set the minutes on the real-time clock. n is in the range of 0 to 59.

MINUTES@
"minutes-fetch"

STACK:

-- n

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

MINUTES@ is used to read the minutes on the real-time clock. n is in the range of 0 to 59.

MOD

STACK:

n1 n2 -- n3

CATEGORY:

83 - Arithmetic

DESCRIPTION:

n3 is the remainder after dividing n1 by the divisor n2. n3 has the same sign as n2 or is zero. An error condition results if the divisor is zero or if the quotient falls outside of the range [-32,768..32,767].

MODIFIERS

STACK:

n -- addr

CATEGORY:

- Memory, LC2/4

DESCRIPTION:

Points to "modifiers" array which is passed to-and-from the OPTOWARE™ driver. This word is used for storing or fetching a value in the "modi- fiers" variable array within the parameter block being pointed at by the "parameters" variable. Since the modifers array is a 2-element array, n determines which element is to be accessed. n is in the range of 0 to 1. The resulting MODIFIERS pointer is equal to the following FORTH expression:

PARAMETERS @ 38 + 2 n * +

MONTHS!
"months-store"

STACK:

n --

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

MONTHS! is used to set the month on the real-time clock. n is in the range of 1 to 12.

MONTHS@
"months-fetch"

STACK:

-- n

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

MONTHS@ is used to read the month on the real-time clock. n is in the range of 1 to 12.

MU/MOD
"m-u-divide-mod"

STACK:

ud1 u1 -- rem quot

CATEGORY:

83 - Arithmetic

DESCRIPTION:

Rem is the remainder and quot is the floor of the quotient after dividing ud1 by the divisor u1. All values and arithmetic are unsigned. MU/MOD is similar to UM/MOD with the exception that the quotient of MU/MOD is a double- precision value.

NAME>
"from-name"

STACK:

addr1 -- addr2

CATEGORY:

- Compiler Word

DESCRIPTION:

addr2 is the compilation address corresponding to the name-field address addr1.

NEGATE

STACK:

n1 -- n2

CATEGORY:

79 - Arithmetic

DESCRIPTION:

n2 is the two's complement of n1; i.e., the difference of zero less n1.

NEXT-LINK
"next-dash-link"

STACK:

-- addr

CATEGORY:

U - System

DESCRIPTION:

NEXT-LINK is a constant for the address of the word NEXT. Addr is the address of NEXT.

NO

STACK:

-- n

CATEGORY:

- Constant, System

DESCRIPTION:

A constant for the SET.SERIAL word for no parity.

NOOP
"no-op"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

A NOOP does nothing.

NOT

STACK:

16b1 -- 16b2

CATEGORY:

83 - Logical

DESCRIPTION:

16b2 is the one's complement of 16b1.

ODD

STACK:

-- n

CATEGORY:

- Constant, System

DESCRIPTION:

A constant for the SET.SERIAL word for odd parity.

OF

STACK:

-- addr n	**(compiling)**
n1 n2 -- n1	**(if no match was found)**
n1 n2 --	**(if match was found)**

CATEGORY:

Comparison - Compiling

DESCRIPTION:

At run time, OF checks to see if n1 is equal to n2. If so, n1 and n2 are dropped and execution continues to the next ENDOF. If n1 is not equal to n2, only n2 is dropped and execution jumps to whatever follows the next ENDOF.

At compile time, OF compiles a comparison, ?BRANCH and reserves space for a jump address. addr is used by ENDOF to resolve the address, n is used for error checking.

OLD

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

OLD is a system variable.

OPTOWARE

STACK:

--

CATEGORY:

- System, Optoware

DESCRIPTION:

This word causes all the parameters in the current parameter-block to be placed on the stack and a call be made to the OPTOWARE™ driver. Upon returning from OPTOWARE™, the parameter-block will be updated with the values which OPTOWARE™ returned on the stack. It is very important that a parameter-block be defined and the parameters variable contain the address of the defined parameter-block before OPTOWARE™ is called.

OR

STACK:

16b1 16b2 -- 16b3

CATEGORY:

79 - Logical

DESCRIPTION:

16b3 is the bit-by-bit inclusive-or of 16b1 with 16b2.

OUT

STACK:

-- addr

CATEGORY:

u - System

DESCRIPTION:

OUT is a user variable which contains a value incremented by EMIT each time EMIT is executed.

OVER

STACK:

16b1 16b2 -- 16b1 16b2 16b1

CATEGORY:

79 - Stack

DESCRIPTION:

Copies of the second stack value to the top of the stack.

P!
"p-store"

STACK:

n port --

CATEGORY:

- Output

DESCRIPTION:

P! outputs the least significant byte of integer n to the 8-bit I/O port specified by port.

P@
"p-fetch"

STACK:

port -- n

CATEGORY:

- Input

DESCRIPTION:

P@ inputs a byte from the I/O port specified and transfers it to the top of the stack. The most significant byte of integer n is 0.

PAD

STACK:

-- addr

CATEGORY:

83 - System

DESCRIPTION:

The lower address of a scratch area used to hold data for intermediate processing. The address or contents of PAD may change and the data may be lost if the address of the next available dictionary location is changed. The minimum capacity of PAD is 84 characters.

PAMWARE (LC4)

STACK:

--

CATEGORY:

- System, LC2/4

DESCRIPTION:

This word causes all the parameters in the current parameter-block to be placed on the stack and a call be made to the PAMUX™ driver. Upon returning from PAMWARE™, the parameter-block will be updated with the values which the PAMUX™ driver returned on the stack. It is very important that a parameter-block be defined and the parameters variable contain the address of the defined parameter-block before OPTOWARE™ is called.

The parameter-block used by PAMWARE™ is the same one used by OPTOWARE™. However, not all of the OPTOWARE™ parameters are used by the PAMWARE™. Only the first location of the POSITIONS array is used by the PAMUX™ driver, therefore, a variable pointer POSITION has been defined.

LC4: This word is only available with LC4.

PARAMETER-BLOCK
"parameter-block"

STACK:

--

CATEGORY:

- Defining, Optoware

DESCRIPTION:

Word used to allocate an area in memory which has a predefined structure for passing parameters to the OPTOWARE™ and PAMWARE™ drivers. Used in the form:

PARAMETER-BLOCK <name>

Refer to page 2-2 for correct usage of a parameter-block structure. Multiple parameter-blocks may be assigned by giving each a unique name. A parameter-block can be accessed after it has been defined by storing its name in the PARAMETERS pointer. See PARAMETERS word for accessing a parameter-block.

PARAMETERS

STACK:

-- addr

CATEGORY:

-- Memory, Optoware

DESCRIPTION:

The PARAMETERS word is used when referring to the "parameters" variable for storing the address of the current parameter-block. Used along with PARAMETER-BLOCK in the form:

```
PARAMETER-BLOCK  BOARD !
                 .
                 .
                 .
BOARD1 PARAMETERS !
```

Any references made to any of the OPTOWARE™ parameters will use the address stored in PARAMETERS as a pointer to the parameter-block specified.

PICK

STACK:

+n -- 16b

CATEGORY:

83 - Stack

DESCRIPTION:

16b is a copy of the +nth stack value, not counting +n itself {0... the number of elements on stack-1}.

0 PICK is equivalent to DUP
1 PICK is equivalent to OVER

POINTER

STACK:

addr --

CATEGORY:

- Defining

DESCRIPTION:

POINTER is a defining word for creating a memory pointer. Used in the form:

addr POINTER <name>

This creates a dictionary entry for <name> **so that when** <name> **is later executed, the address addr is left on the stack.**

POSITION (LC4)

STACK:

-- addr

CATEGORY:

- Memory, LC2/4

DESCRIPTION:

Points to the POSITION variable which is passed to-and-from the PAMUX™ driver. This word is used for storing or fetching a value in the POSITION variable. (POSITION is equivalent to 0 POSITIONS for the OPTOMUX™ driver).

LC4: This word is available only with LC4.

POSITIONS

STACK:

n -- addr

CATEGORY

- Memory, Optoware

DESCRIPTION:

Points to "positions" array which is passed to-and-from the OPTOWARE™ driver. This word is used for storing or fetching a value in the "positions" variable array within the parameter block being pointed at by the "parameters" variable. Since the positions array is a 16-element array, n determines which element is to be accessed. n is in the range of 0 to 15. The resulting POSITIONS pointer is equal to the following FORTH expression:

PARAMETERS @ 6 + 2 n * +

QUERY

STACK:

--

CATEGORY:

- Input

DESCRIPTION:

QUERY accepts a text string from the host communications port. Input will continue up to 80 characters in length until a carriage return (0DH) is encountered. The text string is placed at the address TIB with #TIB equal to the number of characters received and >IN set to 0.

QUIT

STACK:

--

CATEGORY:

79 - Control

DESCRIPTION:

Clears the return stack, sets interpret state, accepts new input from the current input device, and begins text interpretation. No message is displayed.

R>
"r-from"

STACK:

-- 16b

CATEGORY:

C,79 - Stack

DESCRIPTION:

16b is removed from the return stack and transferred to the data stack.

R@
"r-fetch"

STACK:

-- 16b

CATEGORY:

C,79 - Stack

DESCRIPTION:

16b is a copy of the top of the return stack.

R0
"r-zero"

STACK:

-- addr

CATEGORY:

U - System

DESCRIPTION:

R0 is a user variable which contains the initial address of the return stack. It is a 16-bit, single-precision value that is initialized by COLD during system start-up.

RECURSE

STACK:

--

CATEGORY:

I - Control

DESCRIPTION:

RECURSE is used to execute a colon definition recursively. When RECURSE is used in a colon definition, the compilation address of the colon definition is compiled and executed causing the definition to execute itself recursively. Care must be taken to avoid overflowing the stack.

REPEAT

STACK:

--

sys -- (compiling)

CATEGORY:

C,I,79 - Control

DESCRIPTION:

Used in the form:
BEGIN ... flag WHILE ... REPEAT at execution time, REPEAT continues execution to just after the corresponding BEGIN. sys is balanced with its corresponding WHILE. See: BEGIN

ROLL

STACK:

+n --

CATEGORY:

83 - Stack

DESCRIPTION:

The +nth stack value, not counting +n itself is first removed and then transferred to the top of the stack, moving the remaining values into the vacated position. {.. the number of elements on the stack-1} 2 ROLL is equivalent to ROT. 0 ROLL is a null operation.

ROT
"rote"

STACK:

16b1 16b2 16b3 -- 16b2 16b3 16b1

CATEGORY:

79 - Stack

DESCRIPTION:

The top three stack entries are rotated, bringing the deepest to the top.

RP!
"r-p-store"

STACK:

--

CATEGORY:

- System @MINOR HEADING = DESCRIPTION:

RP! initializes the return stack pointer to the address contained in the user variable R0. See R0.

RP@
"r-p-fetch"

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

RP@ returns the return stack pointer address that is present at the time RP@ is invoked and places it on the parameter stack.

S0
"s-zero"

STACK:

-- addr

CATEGORY:

U - System

DESCRIPTION:

S0 is a user variable containing the initial address of the parameter stack. It is a 16-bit, single-precision value that is initialized by COLD during system start-up.

S>D
"s-to-d"

STACK:

n -- d

CATEGORY:

- Conversion

DESCRIPTION:

S>D converts a single-precision number to a double-precision number by sign extending the single-precision number. The high order bit of the single-number is copied into all the high order bits of the double-number.

SECONDS!
"seconds store"

STACK:

n --

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

SECONDS! is used to set the seconds on the real-time clock. n is in the range of 0 to 59.

SECONDS@
"seconds-fetch"

STACK:

-- n

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

SECONDS@ is used to read the seconds on the real-time clock. n is in the range of 0 to 59.

SET.SERIAL

STACK:

n1 n2 n3 -- flag

CATEGORY:

- System

DESCRIPTION:

Initializes the current serial port for number of stop bits, number of dta bits and the parity. n1 is the parity (see ODD, EVEN and NO), n2 is the number of data bits and n3 is the number of stop bits. A TRUE is returned if the port has been initialized, a FALSE is returned if the initialization has failed.

SHIFT

STACK:

w1 n -- w2

CATEGORY:

- Arithmetic

DESCRIPTION:

w2 is the result of shifting w1 n bits. If n is negative, w1 is shifted to the right, if n is positive, w1 is shifted to the left.

SIGN

STACK:

n --

CATEGORY:

79 - Conversion

DESCRIPTION:

If n is negative, an ASCII "-" (minus sign) is appended to the pictured numeric output string. Typically used between <# and #>.

SLEEP

STACK:

n --

CATEGORY:

- System, LC2/4

DESCRIPTION:

The SLEEP word is used to disable a single LC2/LC4 or all LC2/LC4's on a multidropped link which is currently enabled (listening and responding to communications). n represents the unique address of the LC2/LC4 to be disabled. When SLEEP is issued, each LC2/LC4 that is enabled will compare the value on the top of the stack to the value stored in its system variable LC.ADDRESS. If a match is found, that particular LC2/LC4 will be disabled and will not respond to any communication on the host link until an AWAKE word is used with this address. If no value appears on the stack, all LC2/LC4's will be disabled. See AWAKE and LC.ADDRESS.

SMUDGE

STACK:

--

CATEGORY:

- System

DESCRIPTION:

SMUDGE is used during word definition to toggle the "smudge" bit in the length byte of a definition's name field. This prevents an uncompleted definition from being found (via the FIND word) until compiling is completed without an error.

SPACE

STACK:

--

CATEGORY:

79 - Output

DESCRIPTION:

Displays an ASCII space.

SPACES

STACK:

+n --

CATEGORY:

79 - Output

DESCRIPTION:

Displays +n ASCII spaces. Nothing is displayed if +n is zero.

SPAN

STACK:

-- addr

CATEGORY:

U,83 - System

DESCRIPTION:

Span is the address of a variable containing the count of characters actually received and stored by the last execution of EXPECT. See: EXPECT.

SP!
"s-p-store"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

SP! initializes the parameter-stack pointer to the address contained in the user variable S0. See S0.

SP@
"s-p-fetch"

STACK:

-- addr

CATEGORY:

- Stack

DESCRIPTION:

SP@ returns the parameter-stack pointer address that is present at the time SP@ is invoked and places it on the parameter-stack. The following usage:

```
1 2 SP@  @ . . .
```

would result in 2 2 1 being displayed.

To get address of 2nd item on stack, add 2. The following usage:

```
1 2 SP@ 2+ @ . . .
```

would result in 1

2 1 being displayed.

STATE

STACK:

-- addr

CATEGORY:

U,79 - System

DESCRIPTION:

State is a variable containing the compilation state. A non-zero content indicates compilation is occurring, but the value itself is system dependent. A Standard Program may not modify this variable.

SWAP

STACK:

16b1 16b2 -- 16b2 16b1

CATEGORY:

79 - Stack

DESCRIPTION:

The top two stack entries are exchanged.

TASK

STACK:

--

CATEGORY:

- System

DESCRIPTION:

TASK is a definition that is used only as a boundary marker between applications or program segments. The word consists of a colon, a name, and a semicolon to create a no-op definition. It is normally compiled as the first word of an application with all other application words following it in the dictionary. In this manner, the use of FORGET TASK will then forget all of the application.

THEN

STACK:

--
sys -- (compiling)

CATEGORY:

C,I,79 - Control

DESCRIPTION:

Used in the form:

flag IF ... ELSE ... THEN
or
flag IF ... THEN

THEN is the point where execution continues after ELSE, or IF when no ELSE is present. sys is balanced with its corresponding IF or ELSE. See: IF ELSE.

TIB
"t-i-b"

STACK:

-- addr

CATEGORY:

83 - System

DESCRIPTION:

The address of the text input buffer. This buffer is used to hold characters when the input stream is coming from the current input device. The minimum capacity of TIB is 80 characters.

TIME!
"time-store"

STACK:

n1 n2 n3 --

CATAGORY:

- Input/Output, LC2/4

DESCRIPTION:

TIME! is used to set the real-time clock's time. n1 is a number representing the time in hours (0-23), n2 is the minutes (0-59) and n3 is the seconds (0-59).

TIME@
"time-fetch"

STACK:

-- n1 n2 n3

CATAGORY:

- Input/Output, LC2/4

DESCRIPTION:

TIME@ is used to read the real-time clock's time. n1 is a number representing the time in hours (0-23), n2 is the minutes (0-59) and n3 the seconds (0-59).

TOGGLE

STACK:

addr byte --

CATEGORY:

- Logical, Memory

DESCRIPTION:

TOGGLE will Exclusive-Or the byte specified with the contents of the byte at the specified address. This is useful for updating flags in memory.

TRAVERSE

STACK:

addr1 n -- addr2

CATEGORY:

- System

DESCRIPTION:

TRAVERSE is used to move across the name field of a variable-length name- field. addr1 is either the address of the length byte or of the last letter of the name-field. n is a direction indicator. If n=1, the traverse is performed towards high memory. If n=-1, the traverse is performed towards low memory. The resuming address addr2 is the address of the opposite end of the name-field.

TRUE

STACK:

-- -1

CATEGORY:

- Logical

DESCRIPTION:

TRUE is a constant which is equal to the value -1. It is useful for setting flags or logical states to a "true" value.

TX.ENABLE
"transmit-enable"

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

TX.ENABLE is a system variable stored at address addr, that is used by the system to enable or disable output on the currently selected port. Setting TX.ENABLE to -1(true) will enable output. Setting TX.ENABLE to 0(false) will disable any output.

TYPE

STACK:

addr +n --

CATEGORY:

79 - Output

DESCRIPTION:

+n characters are displayed from memory beginning with the character at addr and continuing through consecutive addresses. Nothing is displayed if +n is zero.

U.
"u-dot"

STACK:

u --

CATEGORY:

79 - Output

DESCRIPTION:

u is displayed as an unsigned number in a free-field format.

U.R
"u-dot-r"

STACK:

u width --

CATEGORY:

- Output

DESCRIPTION:

U.R prints an unsigned number right-justified in a field of specified width. If the specified width is less than the number of significant digits, the number is printed without leading blanks. No trailing blanks are printed.

U<
"u-less-than"

STACK:

u1 u2 -- flag

CATEGORY:

83 - Comparison

DESCRIPTION:

flag is true if u1 is less than u2.

U>
"u-greater-than"

STACK:

u1 u2 -- flag

CATEGORY:

- Comparison

DESCRIPTION:

flag is true if u1 is greater than u2.

UM*
"u-m-times"

STACK:

u1 u2 -- ud

CATEGORY:

83 - Arithmetic

DESCRIPTION:

ud is the unsigned product of u1 times u2. All values and arithmetic are unsigned.

UM/MOD
"u-m-divide-mod"

STACK:

ud u1 -- u2 u3

CATEGORY:

83 - Arithmetic

DESCRIPTION:

u2 is the remainder and u3 is the floor of the quotient after dividing ud by u1. All values and arithmetic are unsigned. An error condition results if the divisor is zero or if the quotient lies outside the range {0...65,535}.

UNTIL

STACK:

flag --
sys --

CATEGORY:

C,I,79 - Control

DESCRIPTION:

Used in the form:

BEGIN ... flag UNTIL

Marks the end of a BEGIN-UNTIL loop which will terminate based on flag. If flag is true, the loop is terminated. If flag is false, execution continues to just after the corresponding BEGIN. sys is balanced with its corresponding BEGIN. See: BEGIN.

USER

STACK:

n ---

CATEGORY:

Defining

DESCRIPTION:

A defining word used in the form:

n USER <name>

which creates a user variable <name>. The parameter field of <name> contains n as a fixed offset relative to the user pointer UP for this user variable. When <name> is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable.

VARIABLE

STACK:

--

CATEGORY:

79 - Defining word

DESCRIPTION:

A defining word executed in the form:

VARIABLE <name>

A dictionary entry for <name> **is created and two bytes are ALLOTted in its parameter-field. This parameter-field is to be used for contents of the variable. The application is responsible for initializing the contents of the variable which it creates. When** <name> **is later executed, the address of its parameter-field is placed on the stack.**

VOC-LINK
"voke-link"

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

VOC-LINK is a user variable that contains the address (pointer) of a field in the definition of the most recent vocabulary created. All vocabulary definitions are linked by this field in chronological order via this pointer.

VOCABULARY

STACK:

--

CATEGORY:

83 - Defining word

DESCRIPTION:

A defining word executed in the form:

VOCABULARY <name>

A dictionary entry for <name> **is created which specifies a new ordered list of word definitions. Subsequent execution of** <name> **replaces the first vocabulary in the search order with** <name>**. When** <name> **becomes the compilation vocabulary, new definitions will be appended to** <name>**'s list.**

WHILE

STACK:

flag --

sys1 -- sys2 (compiling)

CATEGORY:

C,I,79 - Control

DESCRIPTION:

Used in the form:

BEGIN ... flag WHILE ... REPEAT

Selects conditional execution based on flag. When flag is true, execution continues to just after the WHILE through to the REPEAT which then continues execution back to just after the BEGIN. When flag is false, execution continues to just after the REPEAT, exiting the control structure. sys1 is balanced with its corresponding BEGIN. sys2 is balanced with its corresponding REPEAT. See: BEGIN.

WIDTH

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

WIDTH is a user variable that contains the maximum number of characters stored in the name-field of a compiled definition. This value is in the range of 1 to 31 characters with 31 as the default. The length byte is not included in the field width specified by width.

WORD

STACK:

char -- addr

CATEGORY:

83 - String

DESCRIPTION:

Generates a counted string by non-destructively accepting characters from the input stream until the delimiting character is encountered or the input stream is exhausted. Leading delimiters are ignored. The entire character string is stored in memory beginning at addr as a sequence of bytes. The string is followed by a blank which is not included in the count. The first byte of the string is the number of characters {0...255}. If the string is longer than 255 characters, the count is unspecified. If the input stream is already exhausted as WORD is called, then a zero length character string will result.

If the delimiter is not found the value of >IN is the size of the input stream. If the delimiter is found >IN is adjusted to indicate the offset to the character following the delimiter. #TIB is unmodified.

The counted string returned by WORD may reside in the "free" dictionary area at HERE or above. Note that the text interpreter may also use this area.

WORDS

STACK:

--

CATEGORY:

- System

DESCRIPTION:

WORDS lists the names of all definitions in the dictionary.

XON0.ENABLE

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

XON0.ENABLE leaves the address of the host xon/xoff enable variable on the stack. A false value in XON0.ENABLE will disable xon/xoff protocol on the host port. A TRUE will enable the xon/xoff protocol.

XON1.ENABLE

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

XON1.ENABLE leaves the address of the com1 xon/xoff enable variable on the stack. A false value in XON1.ENABLE will disable xon/xoff protocol on the com1 port. A TRUE will enable the xon/xoff protocol.

XON2.ENABLE (LC4)

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

XON2.ENABLE leaves the address of the com2 xon/xoff enable variable on the stack. A false value in XON2.ENABLE will disable xon/xoff protocol on the com2 port. A TRUE will enable the xon/xoff protocol.

LC4: This word is only available with LC4.

XON3.ENABLE (LC4)

STACK:

-- addr

CATEGORY:

- System

DESCRIPTION:

XON2.ENABLE leaves the address of the com3 xon/xoff enable variable on the stack. A false value in the XON3.ENABLE will disable xon/xoff protocol on the com3 port. A TRUE will enable the xon/xoff protocol.

This word is only available with LC4.

XOR
"x-or"

STACK:

16b1 16b2 -- 16b3

CATEGORY:

79 - Logical

DESCRIPTION:

16b3 is the bit-by-bit exclusive-or of 16b1 with 16b2.

YEARS!
"years-store"

STACK:

n --

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

YEARS! is used to set the year on the real-time clock. n is in the range 0 to 99.

YEARS@
"years-fetch"

STACK:

-- n

CATEGORY:

- Input/Output, LC2/4

DESCRIPTION:

YEARS@ is used to read the year on the real-time clock. n is in the range 0 to 99.

[
"left-bracket"

STACK:

--

-- (compiling)

CATEGORY:

I,79 - Compiler word

DESCRIPTION:

Sets interpret state. The text from the input stream is subsequently interpreted. For typical usage see LITERAL. See:].

['] "bracket-tick"

STACK:

-- addr
-- (compiling)

CATEGORY:

C,I,83 - Compiler word

DESCRIPTION:

Used in the form:

['] <name>

Compiles the compilation address addr of <name> as a literal. When the colon definition is later executed addr is left on the stack. An error condiiton exists if <name> is not found in the currently active search order. See: LITERAL.

[COMPILE] "bracket-compile"

STACK:

--
-- (compiling)

CATEGORY:

C,I,79 - Compiler word

DESCRIPTION:

Used in the form:

[COMPILE] <name>

Forces compilation of the following word <name>. This allows compilation of an immediate word when it would otherwise have been executed.

] "right-bracket"

STACK:

--

CATEGORY:

79 - Compiler word

DESCRIPTION:

Sets compilation state. The text from the input stream is subsequently compiled. For typical usage see LITERAL. See: [.

\ "backslash"

STACK:

--

CATEGORY:

- System

DESCRIPTION:

Used in the form \ ccccc

Ignore a comment that will be delimited by the end of the line. The \ must be followed by a space.

Appendix

TASK SCHEDULER

```
(       )
(       This is an example of a simple task scheduler)
(       )
(       The resolution of the real-time clock interrupt)
(       on the LC2 is .2 seconds.  This will allow us)
(       to start a new task every .1 seconds.  If a)
(       task takes longer than .2 seconds, other tasks)
(       cannot be performed until the culprit is done.)
(       )
(       Each task is assigned a tick count with a .1 sec)
(       resolution.  A task will be performed every)
(       time its tick count comes around.)
(       )
(       EXAMPLE: a task with a tick count of 3 will)
(       be performed each time the tick)
(       equals 3, 6, 9, 12 ... etc.

:TASK; (  easy forget)

(       QUE will contain the list of tasks and tick)
(       counts for each task)

(       QUE + 0 has the current clock.tick)
(       QUE + 2 has the tick value for the first word)
(       QUE + 4 has the address of the first word)
(       QUE + 6 has the tick value for the second word)
(       .           )
(       .           )
(       .           )
(       QUE + n-1 has address of last word)
(       QUE + n has a zero to indicate the end of que)

VARIABLE QUE 42 ALLOT          (  make space for 10 words)

(       TASK.POINTER will keep track of where you are in)
        (  que)

VARIABLE TASK.POINTER

(       The following group of words are just some simple)
(       tasks to schedule)

:' :'

        58 HOLD     (  stick a : into text buffer)
;

: '/'

        47 HOLD     (  stick a / into text buffer)
;

(       the following word prints the time on the screen in the)
(       HH:MM:SS format)
```

```
(       hours is 3rd on the stack)
(       minutes is 2nd on the stack)
(       seconds is 1st on the stack)


: .TIME
        SWAP ROT                        (  get into correct order)
        0 <#':'###>                     (  convert hours)
        TYPE                            (  type hours)
        0 <#':'###>                     (  convert minutes
        TYPE                            (  type minutes)
        0 <####>                        (  convert seconds)
        TYPE                            (  type seconds)
;
(       the following word prints the data on the screen in the
(       MM/DD/YY format)

(       month is 3rd on the stack)
(       day is 2nd on the stack)
(       year is 1st on the stack)

: .DATE
        SWAP ROT                        (  get into correct order)
        0 <#'/'###>     (  convert months)
        TYPE                            (  type months)
        0<#'/'###>      (  convert days)
        TYPE                            (  type days)
        0<####>         (  convert years)
        TYPE                            (  type years)
;
: TEST
        ."This is just a test" CR
;
: PRINT.TIME
        TIME@                           (  get time from clock chip)
        .TIME                           (  print the time)
        C
;
: PRINT.DATE
        DATE@                           (  get date from clock chip)
        .DATE                           (  print the date)
        CR
;
(       This word is used to stop all the tasks in the queue)
(       It does this by storing a zero in the tick value for)
(       the first word in the que)
(       )
(       the tasks are stopped only if a key press is detected)
```

```
: STOP.RUN
        ?KEY                            (  check for keypress)
        IF
                KEY DROP                (  get key and drop it)
                0 QUE 2+!               (  make que empty)
        THEN
;
: WAIT.FOR.TICK    (  wait for clock tick to change)
        BEGIN                           (  start loop)
                CLOCK.TICK @            (  get current tick value)
                QUE @                   (  get value to compare with)
                =NOT                    (  see if equal)
        UNTIL                           (  if not equal loop)
;


: SEE.IF.MINE                           (  see if tick value is active)
        QUE @                           (  get current tick value)
        TASK.POINTER @ @                (  get next words tick value)
        MOD 0=                          (  get modulo value)
;
: QUE!
        OVER ! 2+
;
(  initialize que)
        QUE 2+                          (  point to first location)
        9 QUE! ' TEST QUE!              (  do TEST every 9 ticks)
        3 QUE! ' PRINT.TIME QUE!        (  do PRINT.TIME every 3 ticks)
        6 QUE! ' PRINT.DATE QUE!        (  do PRINT.DATE every 6 ticks)
        1 QUE! ' STOP.RUN QUE!          (  do STOP.RUN every 1 ticks)
        0 QUE!                          (  stick in que terminator)
```

```
: RUN                                   (  perform words in QUE)

   1 CLOCK.TICK  (  initialize clock.tick)
   BEGIN         (  setup loop)
                 QUE 2+ TASK.POINTER !     (  initialize task pointer)
                 CLOCK.TICK @ DUP          (  get current clock tick)
                 0=IF                      (  if equal to zero)
                   DROP 1 1 CLOCK.TICK !   (  set equal to a 1)
                 THEN
                 QUE !                     (  set que value)
                 WAIT.FOR.TICK             (  wait for the tick to change)
                 TASK.POINTER @ @          (  get first tick value)
                 IF  (  if value isn't zero)
                     BEGIN
                                           SEE.IF.MINE       (  compare que with clock.tick)
                                           IF  (  if value isn't zero)
                                           TASK.POINTER @    (  get a copy of task pointer)
                                           2+ @              (  point to word to execute)
                                           EXECUTE           (  perform word)
                                           THEN
                                           TASK.POINTER @    (  get a copy of task pointer)
                                           2+ 2+ DUP         (  increment pointer)
                                           TASK.POINTER !    (  save updated task pointer)
                                           @ 0=              (  see if tick is equal to zero)
                                           UNTIL
                                           FALSE             (  on stack to continue loop)
                                           ELSE              (  if first item was zero)
                                           TRUE              (  and put a true onto the stack)
                                           THEN              (  to terminate the loop)
                 UNTIL
;
```

SUGGESTED READING

The following is a list of good sources for learning more about FORTH. However, not all of the books listed cover the FORTH-83 standard, but the philosophy remains the same.

Starting FORTH
Leo Brodie
Prentice-Hall Inc., Englewood Cliffs, New Jersey
1981

FORTH Fundamentals, Volume 1, Language Usage
C. Kevin McCabe
Dilithium Press, Beaverton, Oregon
1983

FORTH Programming
Leo J. Scanlon
Howard W. Sams & Co, Indianapolis, Indiana
1982

FORTH Encyclopedia
Mitch Derick & Linda Baker
Mountain View Press Inc, Mountain View, California

Thinking FORTH
Leo Brodie
Prentice-hall Inc, Englewood Cliffs, New Jersey
1984

FORTH-83 STANDARD
A Publication of the FORTH Standards Team
Mountain View Press Inc, Mountain View, California
August 1983

For additional sources refer to:

A Bibliography of FORTH References
David K. Hofert, Editor
The Institute For Applied FORTH Research, Inc.
70 Elmwood Avenue
Rochester, New York 14611

OPTO 22

43044 Business Park Drive • Temecula, CA 92590-3614
Phone: 800/321-OPTO (6786) or 909/695-3000
Fax: 800/832-OPTO (6786) or 909/695-2712
Internet Web site: http://www.opto22.com

Product Support Services:
800/TEK-OPTO (835-6786) or 909/695-3080
Fax: 909/695-3017
E-mail: support@opto22.com
Bulletin Board System (BBS): 909/695-1367
FTP site: ftp.opto22.com